Automated Analysis of Product Disassembly to Determine Environmental Impact

by

David Ikechukwu Agu, B.S

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2009

Automated Analysis of Product Disassembly to Determine Environmental Impact

Approved by
Supervising Committee:

_____

_____

Dedication

To my parents, Vincent and Pamela, my sister, Regina, and my brother, Jeremy.

Acknowledgements

First and foremost I would like to thank God for giving me both the opportunity and the ability to complete this research.

I would like to thank Dr. Matthew Campbell for all of the guidance and support that he has given me over the course of this project. I did not have a research supervising professor when I first arrived at The University of Texas at Austin as a graduate student in 2007 and Dr. Campbell recruited me to work in his lab group while I was enrolled in his graduate-level design optimization course during my first semester. He did his best to match my interests with his field of expertise and the result has been a two-year experience that I have found to be highly rewarding. The work that I have done under Dr. Campbell has been some of the most enjoyable of my entire, nineteen-year academic journey and I would like to express my gratitude to him for steering my work towards its conclusion.

I would like to thank Dr. Richard Crawford for his contributions as the second reader of my thesis.

I would like to thank my fellow lab mates in the Automated Design Laboratory (ADLab) for their support over the past two years.

I would also like to thank the National Science Foundation for the funding of this research through grant DMII-0448806.

I would like to convey my gratitude to Carol Grosvenor for her help with the photography of the products used in my research.

Last, but not least, I would like to thank my family Vincent, Pamela, Regina and Jeremy. I consider the completion of my thesis to be a great personal accomplishment and I would not have come anywhere close to fulfilling this goal without their immeasurable help and support throughout my life.

August 12, 2009

Abstract

Automated Analysis of Product Disassembly to Determine Environmental Impact

David Ikechukwu Agu, MSE

The University of Texas at Austin, 2009

Supervisor: Matthew I. Campbell

Manufacturers are increasingly being held responsible for the fate of their products once they reach their end-of-life phase. This research uses a combination of total disassembly time and recyclability to gauge the environmental impact of a product at this stage of its use. Recyclability, or wasted weight, is a function of the material contained by a product's subassemblies as it is taken apart.

This project suggests a graph-based method of representing product assemblies. Unlike many existing representation methods which are used in the field of automated disassembly, the method proposed here takes component connection methods into account. This, combined with a library of disassembly defining graph grammars, ensures that the disassembly simulation performed on this assembly approximates real-life disassembly procedures as closely as possible.

The results of this simulation are Pareto sets whose contents represent various points in the disassembly process. Each member of the set is evaluated using the two primary parameters of disassembly time and wasted weight. This Pareto set can be used to judge a particular product's performance during end-of-life, from the perspective of recyclability, against that of another product.

Table of Contents

List of Tables

List of Figures

Chapter 1. Background and Introduction

Traditionally, consumers have used and discarded products with little regard for the environmental impact of their actions. Recently however, there have been signs that these practices are changing. Closer attention has been paid to the consumption of raw materials. Since many resources are in increasingly scant supply closer attention must be paid to ensure that the limited reserves which are still available are not used to the point of exhaustion. Additionally disposal practices are also being investigated. Disposal falls into what is called the "end-of-life" phase of a product. In recent years producers have begun to pay more attention to the fate of their products once they reach their end-of-life. In fact, recent legislation, known by various names worldwide and primarily Extended Producer Responsibility (EPR) in the United States, has begun to ensure this change.

The United States currently lags behind other economically developed countries in formally regulating producer responsibility in the end-of-life phase. The European Union and Japan have already enacted government legislation which forces producers to meet certain environmentally-conscious standards with their products. EPR mandates that certain requirements are met before a specific product is allowed to reach the market. The United States has begun to look in this direction however EPR legislation currently exists only at the state level (EPR Laws, 2009). As of July 2009, thirty states have passed anywhere from one to five specific EPR laws but there are no signs that federal legislation will be enacted in the near future (Gutowski, 2007). Moreover, these laws are very diverse and can relate to anything from the recycling of electronics to the regulation of dry-cell batteries. This issue becomes increasingly relevant when considering how much waste the United States produces and the degree to which waste processing is becoming a sizeable industry. For example, in 2006 exports of waste from the United States to China were valued at $6.7 billion (Gross, 2007). In addition to the legislative aspects, producers also cater to increasingly selective customers. Many consumers in the United

States now make selections depending on the relative "environmental friendliness" of one product with respect to another (Environmental Leader, 2007).

Once a product reaches its end-of-life there are two basic options for its treatment. It can either be completely discarded or its materials can be processed in some way for further use. Within this second option there are also many decisions to be made. One choice might be to shred the entire product and separate the different materials afterwards. Alternatively, the product could be disassembled in an ordered manner and components of different material types separated from each other while still intact. Some products have very complex geometries in addition to a large number of components, meaning there may be literally hundreds of different sequences in which a specific product may be disassembled. Each one of these sequences may take a different amount of time and when the labor costs associated with disassembly are taken into account the decision to choose one disassembly sequence over all the others is not trivial. This is where it is useful to take advantage of automated disassembly. Automated disassembly allows a computer to simulate the disassembly process before it is done manually. If used effectively this would also allow the computer to calculate the most efficient disassembly sequence rather than trying to either guess or calculate the times associated with each sequence by hand.

1.1 AUTOMATED DISASSEMBLY

Automated disassembly is not a new concept in engineering and many techniques have been developed over the years to produce the desired results. Each one of these techniques relies on first developing a suitable product representation. This representation may only be used once an automated disassembly algorithm is created. Among the most important information that a product representation can convey is where a specific component is located within the overall assembly. After all, the goal of a disassembly process is not necessarily complete disassembly. For example, if a product is completely made up of steel with only one component being plastic then it would most likely only be necessary to remove the single plastic component before

recycling or further processing could take place. In other instances recycling might not be the final goal. If a product is being serviced and only a few components need to be replaced then it would not make sense to completely disassemble the product. Instead it should only be disassembled until the targeted components are separated. Ideally the product representation should capture enough information to plan for scenarios such as these.

The problem with many product representations is that they do not take into account as much information as is necessary to either simulate the disassembly process accurately or present meaningful results. For example, Srinivasan and Gadh's "wave propagation" approach makes a great deal of effort to establish relationships between each component in an assembly with its neighbors (Srinivasan, 2000). Properties such as "accessibility" and "removal influence" define whether or not a certain component keeps another from being removed from the assembly. One step in the process creates what are called tau and beta waves whose size and intersections determine the sequence of operations necessary to remove a specific component. Figure 1.1 shows a simple product representation using this approach with the circular nodes representing components and the dashed lines representing tau and beta waves. While this approach is effective, the resultant "most efficient" disassembly sequence is presented as the one which requires the smallest number of individual steps. The product representation used in this method simply shows product assemblies as a group of adjacent components without taking into account the various methods which are used to actually connect one component to another. For this reason the assumption that the sequence which takes the smallest number of steps is the most efficient is often incorrect. Due to these different component connection types, a sequence which takes more individual steps than another may actually take less time.

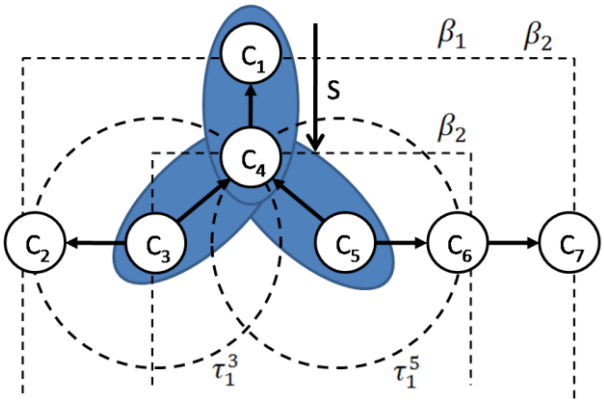

Figure 1.1: Representation of a simple product showing tau and beta wave propagation.

Another approach to analyzing the disassembly problem is the formation of disassembly trees. These trees map all of the possible options which are created by following a certain disassembly sequence. A full disassembly tree would therefore depict every possible sequence which fully disassembles a product. Homem de Mello and Sanderson's AND/OR hypergraph method does this effectively while also combining certain nodes in the disassembly tree together to decrease the time required to search it (Homem de Mello, 1990). In fact, the reduction in the total number of nodes and arcs is one of the main goals of this method. Unfortunately, like the wave propagation method, component connection methods are not taken into account and the main emphasis remains on the number of operations rather than the actual time it would take to complete that particular sequence. Figure 1.2 shows a full AND/OR hypergraph of a simple four-component product.

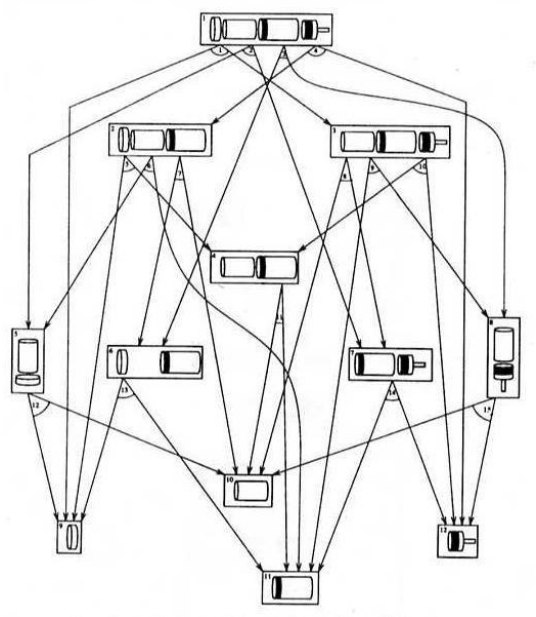

Figure 1.2: AND/OR hypergraph representation of a simple product.

In light of the weaknesses of these existing methods the graph-based representation presented here attempts to take into account the information necessary to accurately calculate the disassembly time of a product. This information includes component dimensions, weight and connection methods among others.

1.2 EVALUATION METHOD

One of the main goals of this research is to combine the aforementioned environmental considerations with a realistic product representation. This requires a parameter to be created which takes these environmental concerns into account. The environmental impact of a product is therefore calculated while keeping two variables in mind: the disassembly time and the wasted weight. The wasted weight is defined as the weight of the components within an assembly which are connected to other components of a different material type. The idea is that a group of components can only be recycled if it is materially homogeneous. When expressed as a percentage of total product weight this wasted weight parameter acts as a gauge for how far along a product is in the disassembly process and therefore how close it is to being able to be completely recycled. Wasted weight also necessitates the inclusion of material type information

in the product representation model. If a disassembly tree can be formed based upon a product representation which takes the necessary information into account then each node in this tree can be evaluated using some combination of these disassembly time and wasted weight parameters to produce the optimal disassembly sequence.

This project makes use of a software tool called GraphSynth which was developed by the Automated Design Lab at The University of Texas for use in automated design projects (GraphSynth, 2009). GraphSynth is written in C# and provides a convenient means for both representing product assemblies and evaluating their disassembly. GraphSynth makes use of graph grammars and graph theory in solving the automated disassembly and analysis problem.

1.3 THESIS OVERVIEW

This document will provide a detailed description of the process used to develop a new product representation, an efficient automated disassembly simulation and an analysis of the disassembly process. It will make use of two similar, competing products and present the results of their disassembly evaluations to show the ability of this method to contrast the environmental impact of one product with another's.

Chapter 2 focuses on the development of the product representation method and presents the representative graphs of the two example products.

Chapter 3 discusses grammar rules and how they are used to simulate the disassembly process.

Chapter 4 describes which component information is necessary to accurately model the disassembly process and how this information is stored in the model. It also introduces the methods which are used to perform disassembly time calculations.

Chapter 5 focuses on the formation of a disassembly tree, how this tree is searched and what information the search process outputs. The section also details the steps that were taken to develop the current version of the tree-searching algorithm.

Chapter 6 returns to the two example products and presents the results of simulations performed on their product representations. This chapter also presents a discussion of the results.

Chapter 7 describes work which can be done to improve the model. The chapter concludes with a summary of the report and a discussion of its potential for use in design and product analysis.

Chapter 2. Product Representation

As was mentioned earlier, the formation of a suitable product representation is vital to accurately modeling the disassembly process. Also, in order to produce meaningful results, it is important to not only model each component and where it is located within the overall assembly but also to include information on how it relates to its neighboring components. A graph-based approach is used to confront this problem.

Within GraphSynth a user is allowed to make use of two types of objects: nodes and arcs. This project has used nodes as representatives of individual components while arcs, which connect nodes to each other, represent the types of connections between components. The arcs are used to simulate a variety of scenarios such as when two components are threaded together or when one component is located on a shaft. Before beginning an in-depth discussion about connection types it is necessary to stress the importance of directionality within product assemblies. For example, a screw can only be removed in one direction and a component which is able to slide along a shaft might only be able to be removed in one direction if the shaft is free on one end but fitted into another component on the other. In light of this, a three-dimensional Cartesian coordinate system is used to define direction-specific features within an assembly. Each of the three axes has dual polarity allowing for a total of six possible directions with respect to the product assembly.

When simulating the disassembly process component connections are vital to determining whether or not a specific component may be removed from the assembly. There are a large number of methods which can be used to connect one component to another and if disassembly is to be accurately modeled then each one of these methods must also be anticipated. In order to build a library of all of these connection types, existing products were taken and disassembled by hand. Notes were taken of the observed connection types and then these were modeled in GraphSynth. Two of the products used were similarly dimensioned leaf blower

models from two different manufacturers: the Toro Model #51586 and the Troy-Bilt TB190BV. These two products are shown in Figure 2.1 and Figure 2.2 respectively.

Figure 2.1: The Toro Model #51586 leaf blower.

Figure 2.2: The Troy-Bilt TB19BV leaf blower.

A second use of directionality in product modeling involves attempting to make the virtual disassembly simulation resemble a real-world disassembly process as closely as possible. When taking example products apart it was often found that disassembly was easier if the product was resting on a surface i.e. a table. However due to the presence of this immovable

surface some operations become impossible, such as removing a screw which is inserted into the side of the product which is in contact with a table. In order to remove this screw it would be necessary to flip the assembly into a new orientation. A blocked surface can be depicted in the product representation through the use of a global variable. If one of the six Cartesian directions is specified as the global label then this places some restrictions on the possible disassembly operations at any point in time. Global labels can be seen in any of the figures showing product assemblies in Section 2.3.

2.1 COMPONENT CONNECTION TYPES

In GraphSynth nodes and arcs are capable of holding a variety of information. Arcs, for instance, may be given names, labels or variables. The type of connection is stored within the label field. Figure 2.3 shows the window within GraphSynth which may be used to edit information about a specific arc. Two components may have more than one type of connective relationship between them and multiple connection types can be stored on one arc as separate labels. The connection types or constraints which have been discovered so far can be separated into the following categories: rectangular constraints, radial constraints, threaded fasteners, press fits, rivets, adhesives and windings. An additional connection which is not necessarily physical and will be explained later on is the virtual rectangular constraint.

Figure 2.3: The arc properties window within GraphSynth.

2.1.1 Rectangular Constraints

Rectangular constraints are the most common connection type found within the investigated products. A rectangular constraint between two components exists when, due to the positioning of one component with respect to another, one of the components cannot be removed from the entire assembly if translated in a specific direction. The left-hand side of Figure 2.4 shows how a rectangular constraint is depicted in GraphSynth. The right-hand side shows an example of two-components which share such a relationship.

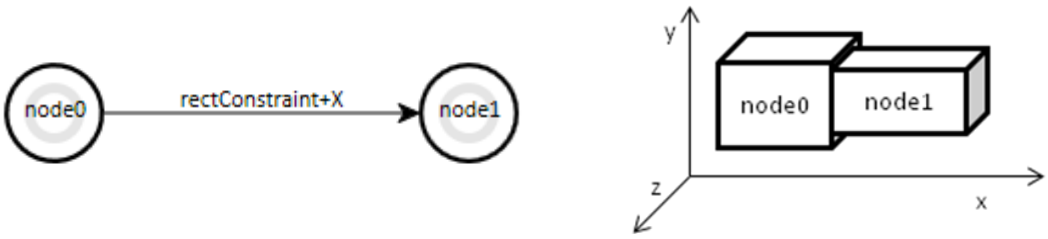

Figure 2.4: The graphical depiction of a rectangular constraint within GraphSynth (left) and a three-dimensional depiction of the same situation (right).

The meaning of the graphical representation is simple. Using the nomenclature in the figure above, node0 is constrained by node1 in the positive X direction by a rectangular constraint. In other words, due to the object signified by node1, node0 cannot be translated out of the assembly in the positive X direction. There are additional points of note related to the figure. The word above the arc, rectConstraint+X, is the label which describes the connection represented by the arc. The arrowhead on the arc signifies which of the two components is constrained by the other using the connection specified by the arc label. In this scenario it would also follow that node1 is constrained by node0 by a rectangular constraint in the negative X direction. Rectangular constraints are one of many types of constraints which exist as pairs of arcs between two specific components. However when modeling an assembly in GraphSynth it is not necessary for a user to draw and label both arcs in the pair. As long as one of these two arcs is drawn and labeled correctly, a "fixer rule" will automatically draw and label the second arc before the disassembly simulation begins. Fixer rules, and rules in general, will be explained in Chapter 3. As can be seen from the directional suffix, +X, there are six types of rectangular constraints, two along each of the three Cartesian axes. The labels associated with these six constraints are: rectConstraint+X, rectConstraint-X rectConstraint+Y, etc.

2.1.2 Radial Constraints

The radial constraint group is somewhat similar to that of rectangular constraints. This group of three constraints is used when one component circumferentially surrounds another. Figure 2.5 shows both a graphical and three-dimensional representation of a radial constraint.

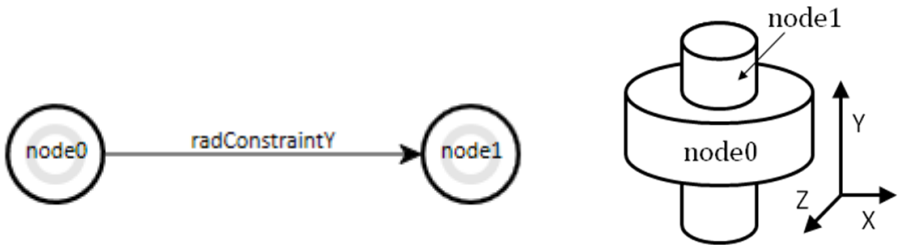

Figure 2.5: The graphical depiction of a radial constraint within GraphSynth (left) and a three-dimensional depiction of the same situation (right).

Unlike rectangular constraints there are only three types of radial constraints. It is seen from the diagram that the labels associated with radial constraints are radConstraintX, radConstraintY and radConstraintZ. While the arrowhead in the figure still signifies that node0 is constrained by node1 with the constraint specified, the directional suffix indicates the axial direction of the "shaft" component. This means that unlike rectangular constraints the directional suffix indicates the one free direction. Alternatively an arc could be drawn from node0 to node1 and carry four rectangular constraint labels with directional suffixes of +X, -X, +Z and –Z. This label assignment would have the same meaning as the one shown in the figure but shows how radial constraints can be used to simplify the representation process. Radial constraints also always exists in pairs however both of their directional suffixes are the same. Since the axial direction does not require polarity and no radial direction information is specified directly, the direction of the radial constraint placed on node1 by node0 is the same as the one placed on node0 by node1. As was the case with rectangular constraints, only one of the two arcs need be modeled since a fixer rule will add the second with the appropriate label. In the situation depicted in Figure 2.5 there is an alternate graphical representation that could be used which shows another relationship between rectangular and radial constraints.

2.1.3 Threaded Fasteners

The constraints associated with threaded fasteners are the first of three groups of constraints which do not require one pair of arcs for each pair of connected components; these

constraints only require one arc. In general, the threaded arc originates from the female threaded component and terminates at the male threaded component (with origin and termination sites defined by the end of the arc on which the arrowhead is placed). This usually means that the arc will point towards components which represent bolts or screws. Likewise, the directional suffix on the arc represents the direction of screw or bolt removal. An exception occurs when there is a male threaded component which is simply threaded but tools are applied to the male threaded component for disassembly purposes. An example of this is a situation where there is a simple threaded shaft with a nut attached to it. In this case the arc would originate at the shaft and terminate at the nut. This is done for reasons involving input variables and disassembly time calculation (Chapter 4). There is one threaded constraint associated with each of the six Cartesian directions resulting in a set of constraints with labels of: thread+X, thread-X, thread+Y, etc. Figure 2.6 depicts the graphical representation of a thread constraint between two components.

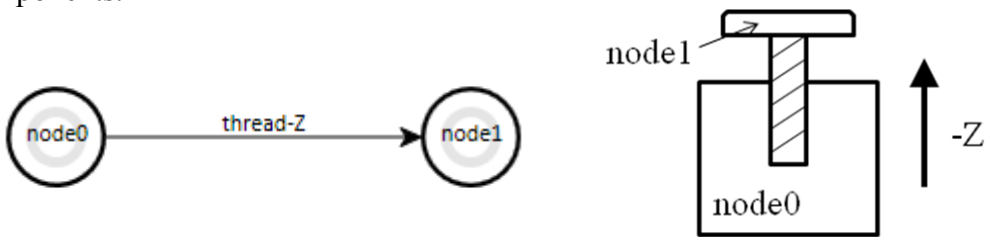

Figure 2.6: The graphical depiction of a thread constraint within GraphSynth (left) and a depiction of the same situation (right).

2.1.4 Press Fits and Rivets

Press fits are very similar to rectangular constraints. Like rectangular constraints there are six variations (one for each Cartesian direction) and they exist in pairs. The meaning behind the graphical representation is very simple. The component from which the arc originates is press-fitted into the component at which the arc terminates, in the direction specified by the arc label's directional suffix. The six press fit label variations are pressFit+X, pressFit-X, pressFit+Y, etc. The second arc in the press fit pair is drawn by a fixer rule if the user chooses not to do so and the label which is added simply has a directional suffix along the same axis as the user-defined

arc but with opposite polarity. Rivets are identical to press fits in their representation and in the meaning associated with the arc itself. The only difference of course is in the arc labels which, in the case of rivets, can be labeled with rivet+X, rivet-X, rivet+Y, etc.

2.1.5 Adhesives

The adhesive "group" is unique in that there is no direction information and thus only one variation. The only label attributed to this constraint is simply "adhesive." It is also unique in the fact that the adhesive arc is doubly directed – that is, there are arrowheads on both ends of the arc. The doubly directed arc is simply placed between two components which are bonded together using an adhesive substance. If there are two components, node0 and node1, which are fastened together with adhesive, their graphical representation would appear as is shown in Figure 2.7.

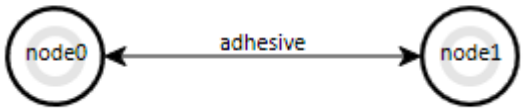

Figure 2.7: The graphical depiction of the adhesive constraint within GraphSynth.

2.1.6 Windings

The last group of constraints is used in instances where one component is wound around another. These arcs also do not exist in pairs. The node from which the arc originates is the component which is wound while the node at which the arc terminates is the spool. Like radial constraints, the directional suffixes on the winding constraint arcs indicate axial directions. In this case this is the Cartesian axis along which the spool lies. Since axial polarity is not an issue there are three labels which can be attributed to winding constraint arcs: windingX, windingY and windingZ. As was mentioned earlier, one arc can possess combinations of arc labels. If for some reason there was a component which was completely encased in wire for instance an arc

could be drawn from the wire to the encased component and it would be labeled with all three of the winding label possibilities.

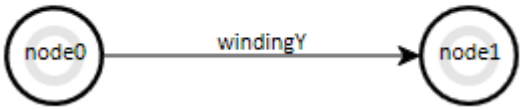

Figure 2.8: The graphical depiction of a winding constraint within GraphSynth.

2.1.7 Virtual Rectangular Constraints

Virtual rectangular constraints were created as a preventative measure for situations in which components suddenly become underconstrained as a result of the removal of another component. A simple scenario in which this may occur is when there are three stacked plates. Since the bottom and middle plates are in contact then there would be a regular rectangular constraint between them and likewise for the middle and top plates. However if the middle plate were able to be removed without disturbing the others then there would now be a rectangular constraint between the bottom and top plates. This can be planned for if seen in advance by the user however the virtual rectangular constraints offer a way around this. Arcs carrying these labels are created automatically by fixer rules, not by the user. For the most part this group is identical to that of regular rectangular constraints. The arcs exist in pairs and there are six variations. The meaning associated with the directionality of the arc is also identical to that of rectangular constraints. The labels assigned to virtual rectangular constraint arcs are as follows; virtualRectConstraint+X, virtualRectConstraint-X, virtualRectConstraint+Y, etc.

These are the types of constraints which have been found so far through investigation. The investigation of more products will likely lead to the discovery of more connection types however for the products which have been used over the course of this project the preceding constraints have been sufficient. Additionally considering the connection types covered in the literature of assembly modeling, we believe the current set covers a substantially large number of products.

2.2 MODULES

Modules originally arose as a concept to allow the user to choose whether or not certain objects within a product assembly are treated as a single component or as a collection of components. However their inclusion has yielded other benefits as well. An example of a module is the motor which is housed inside a leaf blower. The motor may be thought of as a singular component itself however it also contains other components such as screws, washers, an impeller, etc. which are made of many different materials. When disassembling a product it would not make sense to begin taking the motor apart until it is completely separated from the other components. If this is the case then the motor can be represented by a special node, attributed with a "module" label. The graph of the motor's individual components is then created in a separate file. The name of this file is entered as the second label of the module node. By typing the containing folder's directory into the leaf blower graph's "name" field the module node which represents the motor is replaced by the motor component graph once a special module rule is applied. The workings of the module rule as well as a visual depiction of its effect are given in Section 3.2.4.

2.3 EXAMPLE PRODUCTS

The following figures depict the two example products of interest for this document. There are three figures for each of the products. In each case, the first figure shows an "exploded view" photo. In both photos the motor remains fully assembled. The second figure for each product depicts the assembly with a module node (which is highlighted in red), meaning it depicts the motor as a single component, while the third figure for each product depicts the graph of the motor itself. The type of graph depicted in the second figure will be referred to as the simple representation from this point on.

2.3.1 Toro Model #51586 Leaf Blower

Figure 2.9: Exploded view photo of the Toro leaf blower.

Take note of the names and labels of the nodes and arcs in Figure 2.10. Labels for both arcs and nodes are enclosed by parentheses. Node names serve no function other than to make the graph more understandable to the user. They do not make any contribution to the disassembly simulation or any calculations. In this way the module node is unique in that it is the only node which carries a label with any real meaning. To reiterate, the first label simply designates the node as a module while the second gives the file name of the graph which represents the module's individual components. Also take note of the –Z global label in the top left-hand corner of the image.

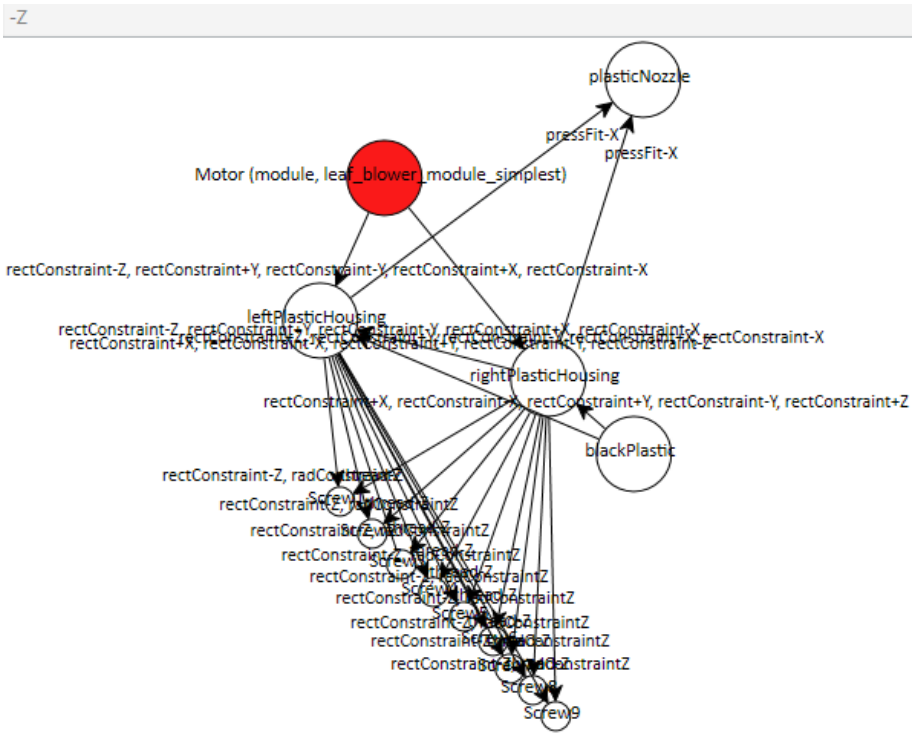

Figure 2.10: Simple representation of the Toro leaf blower in GraphSynth.

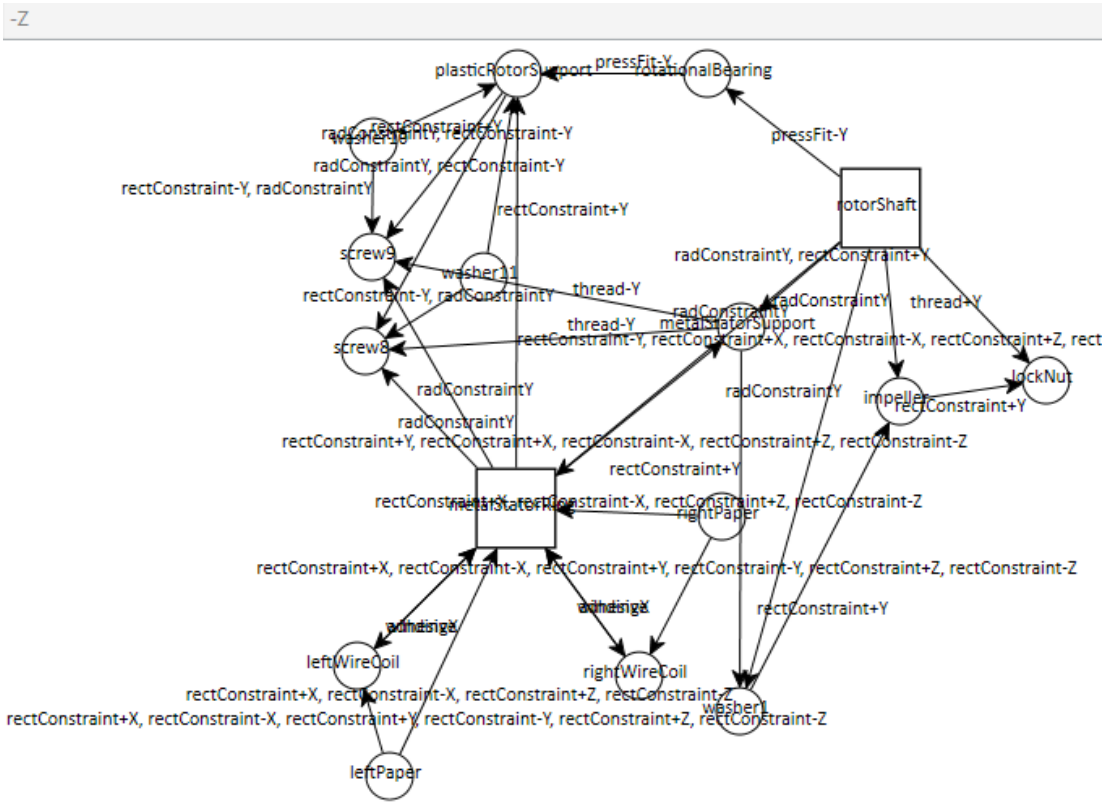

Figure 2.11: Representation of the Toro motor in GraphSynth.

2.3.2 Troy-Bilt TB190BV

Figure 2.12: Exploded view photo of the Troy-Bilt leaf blower.

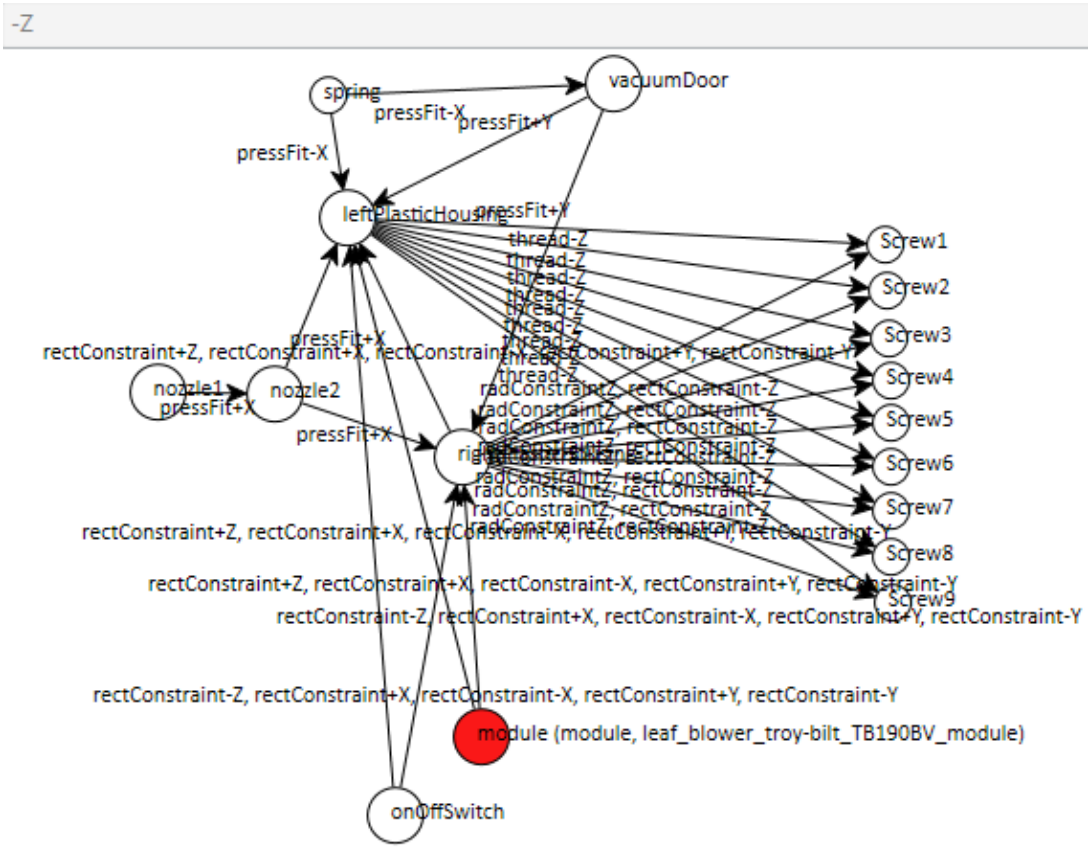

Figure 2.13: Simple representation of the Troy-Bilt leaf blower in GraphSynth.

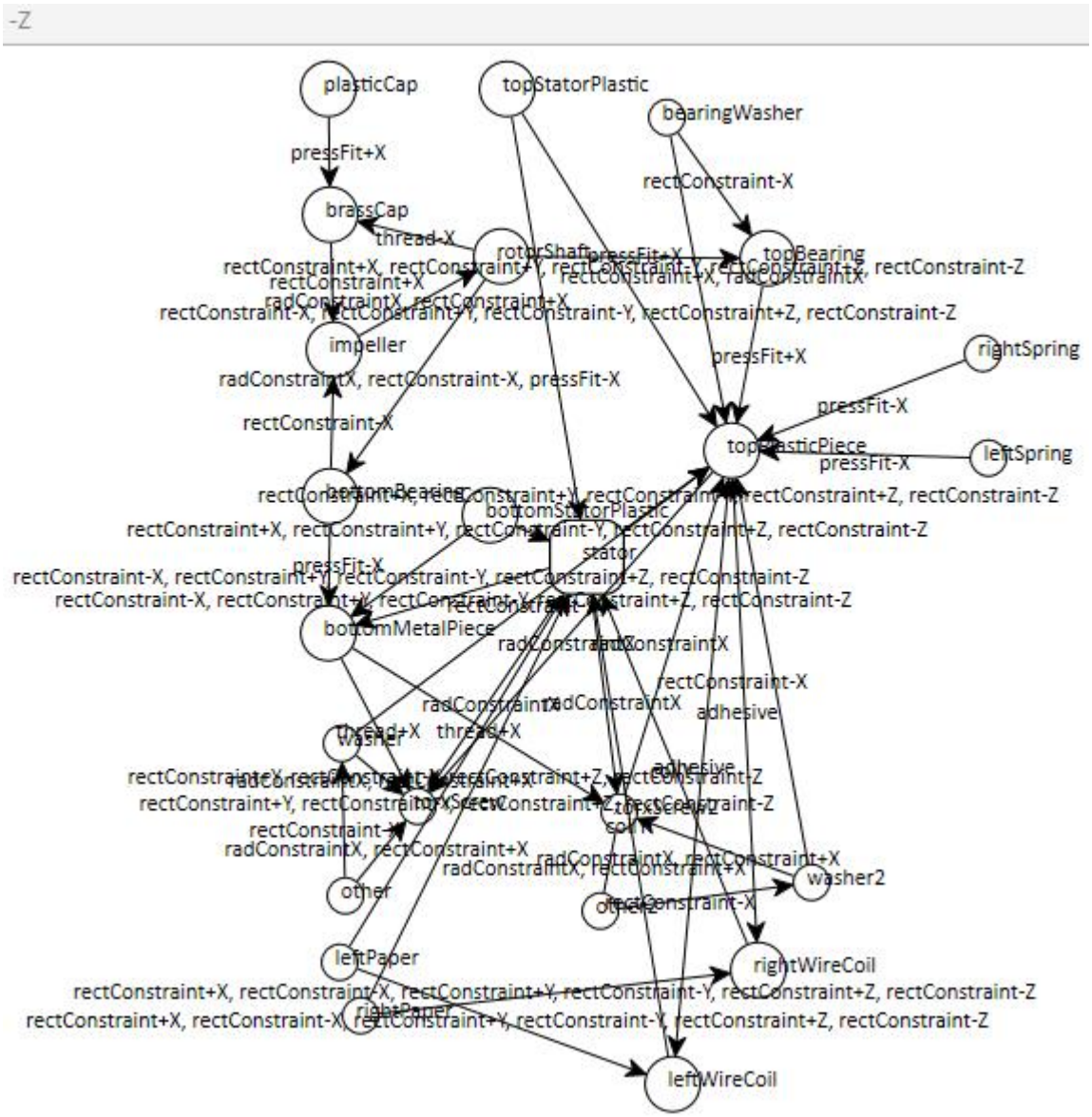

Figure 2.14: Representation of the Troy-Bilt motor in GraphSynth.

Chapter 3. Grammar Rules

A library of connection types which are used to build a graphical representation of a product cannot be used to simulate disassembly unless information about the potential removal of these connections is defined. Grammar rules are used for this very purpose. For example, if there is a bolt screwed into a flat plate and it is removed in the positive Z direction, there must be a rule which specifies when this connection can be removed. After all, if this plate and bolt combination is inside of some housing, then the housing must be removed prior to removing the bolt. Grammar rules should be able to govern the removal of each of the constraints listed in the previous chapter.

3.1 Rule Recognition and Application

3.1.1 L and R

Grammar rules in GraphSynth hold two important pieces of information: recognition and application. The recognition aspect of a grammar rule searches for locations in a graph where the rule can be applied. The application aspect specifies what happens to the graph once the rule is applied. Grammar rules possess two subgraphs a left-hand side (L), reserved for recognition purposes, and a right-hand side (R), for rule application. Recognition has two levels of complexity. First, the subgraph housed in L can be visually matched with an identical location in the assembly or "host" graph. If matching does occur a parametric recognition function can be used to provide extra requirements for rule recognition. These extra requirements can include any number of visual or non-visual graph properties such as specific node labels. In the same way, R hosts a subgraph which shows the result of rule application. If the requirements specified by the left-hand side of the rule are met then the location in the graph where L was identified is modified to look like the subgraph shown in R. There is also the potential for parametric application functions which provide additional information concerning the rule's application. Figure 3.1 shows a grammar rule window as it appears in GraphSynth. Additionally, global labels can be specified which negate the recognition of a particular rule. For example if +X is

chosen as a negating label then for an assembly graph that carries a +X global label the rule in question will not be recognized.

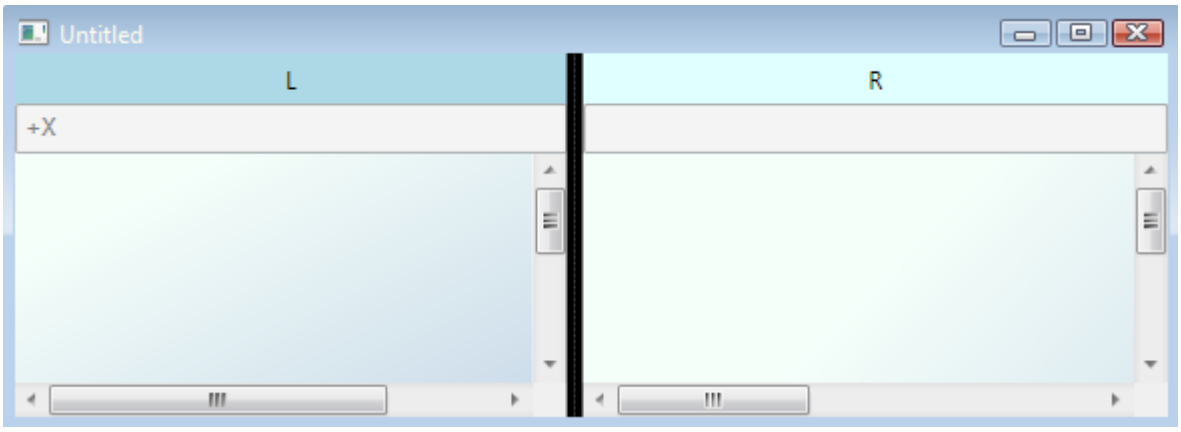

Figure 3.1: The grammar rule window in GraphSynth.

3.1.2 Parametric Functions

Parametric functions can be used to provide extra information to both the recognition and application sides of rules. This information may not be able to be captured visually or may require the investigation of arcs and nodes not depicted in the L and R subgraphs. Parametric functions are created and edited in a C# editor and can be used to directly access the assembly graph and the rule itself. These functions are more easily understood with examples. For examples of both recognition and application parametric functions please see the individual rules described in the next section.

3.2 Rule Sets

Rules do not only have to specify information concerning the separation of components. Owing to this, the rules have been grouped together into rule sets depending on their function. Four rule sets (numbered zero through three), containing forty-one total rules, have been created: the fixer rule set, the disassembly rule set, the flip rule set and the module rule set. The fixer rule set contains three rules which make the product graph creation process simpler and more accurate. A secondary function is that they search for information which appears to have been omitted from the graph by the user by mistake. The disassembly rule set contains thirty-one rules

which define actual disassembly steps, such as removing a screw. The flip rule set contains six rules which are used when the assembly must be flipped over in order for disassembly to continue. The module rule set contains a single rule which is used when the disassembly of a module is to commence. Prior to the application of this rule any modules are treated as singular components.

To accurately simulate the disassembly of a product, rule sets must interact with each other in a certain way. A flowchart depicting the order of rule set interaction is shown in Figure 3.2. First, fixer rules must be applied to a product graph to ensure that all of the necessary arc pairs have been drawn and labeled correctly. Rules from this group are continually applied until no more are recognized. After this, rule set #1 is entered and the disassembly rules applied. If nothing else can be done rule set #2 is entered to see if flipping the assembly into a different orientation would allow more disassembly rules to be applied. If this is the case one flip rule is applied and the process reenters rule set #1. If not, rule set #3 is entered where the graph is then checked for modules. If there are no modules in the graph then the process terminates at this point. If a module is identified then the graph which represents the module components is copied into the simple representation of the product and the node which represents the module as a single component is removed. The process then reenters rule set #0 and fixer rules are applied to the newly added nodes and arcs, if necessary.

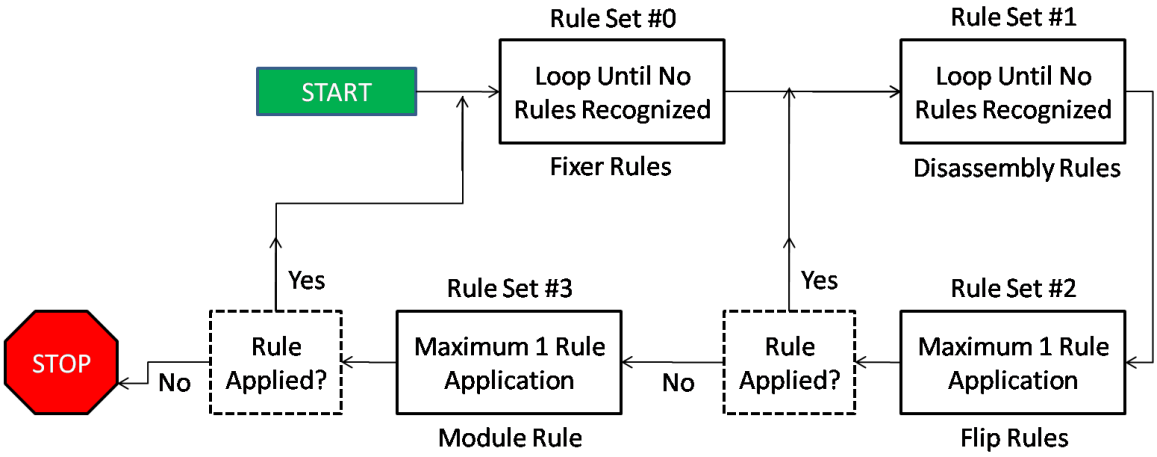

Figure 3.2: Flowchart depicting rule set interaction.

The grammar rules themselves are fairly complex. The subsequent sections will briefly describe the rules in each set while more detailed descriptions of specific rules can be found in Appendices A through D.

3.2.1 Rule Set #0: Fixer Rules

The fixer rule set contains three rules which are used in three very different scenarios for distinct purposes. Once these three rules have been applied to the assembly graph, the product representation should be complete and be ready for a disassembly simulation. The first, called the single fixer rule, is intended to simplify the user's graph creation process by automatically adding the arcs and arc labels required for constraints which exist in pairs. The double fixer rule is used in other situations where it is determined that the user has forgotten to add a label when creating the assembly graph. The cascade fixer rule is responsible for adding virtual rectangular constraints. In-depth descriptions of each of the three fixer rules can be found in Appendix A.

3.2.2 Rule Set #1: Disassembly Rules

Each rule in the disassembly rule set corresponds to the removal of one of the connection types described in Section 2.1. For the most part the left and right-hand sides of these grammar rules are identical. This means that all of the work is done by the recognition and application parametric functions. In fact there is only one rule in this set, the adhesive rule, which does not have identical L and R subgraphs. Each group of connection types has a unique set of recognition conditions and application consequences specified by their respective parametric functions. Each disassembly rule group is explained in Appendix B.

3.2.3 Rule Set #2: Flip Rules

The flip rule set is entered once no more disassembly rules can be applied. This situation may arise for two reasons. First, there may simply be no more possible disassembly operations. This means the product may already be completely separated into groups of components which are materially homogeneous. Secondly, there may still be disassembly operations left to perform however, given the orientation of the remaining components with respect to an immovable

surface these operations cannot be performed. The flip rules are used if and when this second scenario arises. The effect of flip rules is described in more detail in Appendix C.

3.2.4 Rule Set #3: Module Rule

The module rule set consists of a single rule. The only condition for the recognition of this rule is that there is at least one node in the host graph that carries the "module" label. This signifies that a component which has been designated a module still exists and therefore further disassembly can take place. The rule imports the assembly graph which shows the individual components of the module into the graph of the rest of the product. For details on how the rule accomplishes this and an image displaying its effect, refer to Appendix D.

Chapter 4. Input Variables and Disassembly Calculations

The ultimate goal of this project is not simply to simulate product disassembly but to return important information relating to it. This output information has varying degrees of accuracy depending on the accuracy of the information input by the user. None of the desired information is necessary to output information about the disassembly process since default values are assumed in the absence of input from the user. However this results in potentially inaccurate assumptions such as product assemblies that are materially homogeneous. While some assumptions are more damaging to the simulation's accuracy than others, as much information about individual components should always be provided as is possible.

Each type of information can be stored as a single numerical input variable. Depending on the value of the variable and where it is stored it can take on many interpretations. All input information is stored in the "variable" fields of nodes and arcs. These fields hold lists of comma delimited variables and each position in the list holds a predetermined type of information. When a simulation is run and a certain grammar rule is applied, it will look for information in a specific node or arc variable position to perform calculations relating to disassembly time and wasted weight. Node variables always provide information about specific components and arc variables provide information about the physical connections between components.

4.1 NODE VARIABLES

Each node in an assembly, regardless of the type of component it represents, has the potential to store the same type of information. Since each rule will only use information in specific variable positions any nonsensical information will never be used. For example the thread removal rule will result in the request for access to the node variable position designated to hold thread length information. If information is stored in this position for a component which is not threaded it will never be used since it should not have a threaded constraint relationship with any other nodes in the graph. Overall there are ten variable positions which can hold meaningful information for each component in an assembly.

4.1.1 Node Variable 0: Material Information

For nodes, variable position 0 is one of two which should be considered mandatory input information from the user. The absence of material information leads to the aforementioned assumption of a materially homogeneous assembly. If this were the case then no disassembly should be required since the entire product could be recycled as is.

The current list of possible materials is not very large as it is limited to the materials found in the products investigated so far. Further specificity can also be provided to materials such as steel or plastic however the current level of detail is sufficient to produce meaningful results. Table 4.1 shows the eight possible values for the material variable and the meaning of each option.

Variable Value	*Material*
0	No material
1	Plastic
2	Aluminum
3	Copper
4	Steel
5	Iron
6	Other Metal
7	Other Non-Metal

Table 4.1: Possible values for node variable 0 and their meanings.

4.1.2 Node Position 1: Weight

Node position 1 holds the weight of each component. This information is essential when calculating wasted weight and sometimes rectangular and radial constraint removal as well as the time related to flip rule operations. The calculations for these latter two situations require weights measured in pounds-force therefore the value stored in this position must be in the correct units to ensure correct output information. Like the material variable, weight input should be considered mandatory. Weight is one of the only two pieces of information that the wasted weight calculation uses and unless it is specified a default value of zero is used.

4.1.3 Node Positions 2 and 3: Geometric Dimensions

Node positions 2 and 3 necessitate the introduction of the source of many of the values used for disassembly calculations. Geoffrey Boothroyd and Peter Dewhurst are two engineers who have done extensive work in the field of design for assembly. Some of this work includes valuations for the time relating to many common assembly operations. The results of some of their research are used in this project for time calculations relating to disassembly operations. Many of these calculations require information about the geometric dimensions of the components being used in the current operation. Boothroyd and Dewhurst defined two geometric parameters called thickness and size which provide sufficient information for many time calculations. Thickness is defined as "the length of the shortest side of the smallest rectangular prism which encloses the part." Size is defined as the "length of the longest side of the smallest rectangular prism that can enclose the part" (Boothroyd, 1989). The unit of measurement for these dimensions is the millimeter. When used together with the handling information stored in variable positions 4 through 7, calculations relating to rectangular and radial constraints can be performed.

4.1.4 Node Positions 4 through 7: Handling Information

Variable positions 4 through 7 relate to "manual handling," a term which Boothroyd and Dewhurst define as "the grasping, transportation and orientation of parts or sub-assemblies before they are inserted into or added to the work fixture or partially built-up assembly." Boothroyd and Dewhurst's original research dealt with the problem of orientation, in which a part had to be lined up with its site of subsequent insertion or fastening; in most cases this is not an issue for disassembly processes. The table which the two authors developed is greatly reduced due to this fact. Table 4.2 shows the meaning for variables in position 4. In this table "large size" is defined as a weight greater than 10 lbs.

Variable Value	*Meaning*
1	Only one hand needed for manipulation
2	One hand needed with grasping aids
3	Two hands are needed for manipulation
4	Two hands are needed due to large size

Table 4.2: Possible values for node variable 4 and their meanings.

Combined with the meanings listed in Table 4.2, Table 4.3 gives the meanings of all possible combinations of variables which represent handling information.

If variable 4 equals 1		
Var. 4	*Var. 5*	*Meaning*
1	1	One hand needed for manipulation, parts are easy to grasp and manipulate
1	2	One hand needed for manipulation, parts present handling difficulties

If variable 4 equals 2				
Var. 4	*Var. 5*	*Var. 6*	*Var. 7*	*Meaning*
2	1	1	1	Tweezers needed, optical magnification not needed, easy to grasp/manipulate
2	1	1	2	Tweezers needed, optical magnification not needed, difficult to grasp/manipulate
2	1	2	1	Tweezers needed, optical magnification needed, easy to grasp/manipulate
2	1	2	2	Tweezers needed, optical magnification needed, difficult to grasp/manipulate
2	2			Parts need standard tools other than tweezers
2	3			Parts need special tools

If variable 4 equals 3		
Var. 4	*Var. 5*	*Meaning*
3	1	Parts present no extra handling difficulties
3	2	Parts present handling difficulties

If variable 4 equals 4				
Var. 4	*Var. 5*	*Var. 6*	*Var. 7*	*Meaning*
4	1	1	1	One person with no mechanical assistance, parts do not nest/tangle, parts are hard to grasp
4	1	1	2	One person with no mechanical assistance, parts do not nest/tangle, parts are easy to grasp
4	1	2		One person with no mechanical assistance, parts severely nest/tangle/are slippery
4	2			Parts need special tools for grasping/manipulation

Table 4.3: Meanings of combinations of variables in positions 4 through 7.

4.1.5 Node Positions 8 and 9: Thread Information

The final two node variable positions are only used when a component is being unscrewed. Most of the components discovered so far have been either bolts or screws however information from these positions is also needed for calculations pertaining to nuts. Three pieces of information are required to perform time calculations related to unscrewing:

1. Power source: is the removal tool human powered or power assisted?

2. In the cases of screws and bolts, what is the head type?

3. What is the thread length through which the component must be turned?

Boothroyd and Dewhurst have also done research into the insertion of threaded components (Boothroyd, 2002). The times associated with screw insertion and removal have been assumed to be identical if the answers to the above three questions are the same in both cases. The combined answers to questions 1 and 2 are represented by variable 8 while the numerical value which answers question three (and measured in inches) is stored in position 9. Table 4.4 provides the meaning of each of the possible variable values for position 8.

Variable Value	*Meaning*
0	No head type
1	Human-powered removal, slotted head
2	Human-powered removal, Phillips head
3	Human-powered removal, hex head
4	Power-assisted removal, Phillips head
5	Power-assisted removal, slotted head

Table 4.4: Meanings of variables stored in position 8.

4.2 ARC VARIABLES

Unlike nodes, only certain types of arcs accept input variables. All of these arcs involve connections between components which are more secure than mere contact; this excludes connections in the rectangular, radial and virtual rectangular constraint groups. This means that arcs which carry labels from the press fit, adhesive and winding groups can take variable inputs. Rivet constraints do not take input information because currently only one time value is used for rivet removal.

The range of values for input variables is much smaller in arcs than in nodes. In the case of press fit constraints the lack of variation is explained by the lack of variation in the connections themselves from one case to the next. There are only two possible variable values for press fit removal, zero and one, which again correspond to time values ascertained by Boothroyd and Dewhurst. These two values correspond to the rather subjective difficulty of press

fit removal. Easier removal is given a value of zero and more difficult removal a value of one. In the case of winding and adhesive constraints this is due to the limited information which has been gathered thus far regarding these connections' removal. Ideally a library would be created of various adhesives in a list similar to that of component material type. Each adhesive could then have a solvation time associated with it, if solvent is actually needed to separate the two components rather than manual separation, and time calculations performed using these values. At the current juncture however this information has not been added and the single variable input into an adhesive arc's variable field corresponds to an estimate of the time required to forcefully separate the two components; this method of removal has been sufficient in the products investigated thus far. Likewise no information has been found regarding unwinding time. It has been assumed to be some function of the coil diameter and the number of turns. At present winding arcs accept these two values in variable positions 0 and 1 respectively.

4.3 DISASSEMBLY TIME CALCULATIONS

If all potential information is provided for the components and relevant connections in the assembly representation, very accurate calculations can be made. These calculations relate to the two parameters of disassembly time and wasted weight. Disassembly time varies according to the type of connection being broken. Some time values are determined using equations which are functions of input variables. Others require simple matching up of combinations of variables with time values stored within hard-coded functions. Each rule in the disassembly and flip rule sets involves physical manipulation of the product assembly therefore each rule must have a time associated with its application.

4.3.1 Contact Constraint Removal Time

The contact constraints are those which belong to the rectangular, radial and virtual rectangular constraint groups. All that is needed to calculate the disassembly time relating to their removal are the four variables in node positions 4 through 7 and rarely the component

weight. Each combination of handling variables has a specific disassembly time which it corresponds to and these variables are stored inside of a hard-coded function.

4.3.2 Press Fit and Rivet Removal Time

The removal of press fit constraints also involves the simple matching of variables with a corresponding disassembly time within an existing function. Similarly, the single time associated with the removal of rivet constraints is also stored within a function.

4.3.3 Thread Removal Time

Thread removal time calculations actually involve a set of linear equations developed by Boothroyd and Dewhurst. For variables with values ranging from one to five in variable position 8, the following set of equations (4.1) through (4.5) is used. In each equation l represents the thread length and is measured in inches.

$$time\ (s) = 1.75 + \frac{13.25}{14} \cdot l \qquad (4.1)$$

$$time\ (s) = 3 + \frac{6}{14} \cdot l \qquad (4.2)$$

$$time\ (s) = 2.5 + \frac{2.5}{14} \cdot l \qquad (4.3)$$

$$time\ (s) = 1.75 + \frac{1.25}{14} \cdot l \qquad (4.4)$$

$$time\ (s) = 1.5 + \frac{1.25}{14} \cdot l \qquad (4.5)$$

4.3.4 Adhesive Removal Time

Adhesive removal time determination was touched on earlier. The single, user-defined input variable designated to adhesive constraint arcs corresponds to an approximation of the time required for separation of the two components.

4.3.5 Flip Time

Calculations for the flip rule set involve checking for similar weight and handling characteristics among the components which are to be flipped over. The time required to flip the subassembly is taken to be the same as the time required to "manually handle" a component of similar weight and handling characteristics. If a single one of the components in the subassembly requires two hands for manipulation due to large size then it is assumed that the entire subassembly is also of a large size and the corresponding handling time for a component of this size is used. An identical assumption is made if any of the components requires two hands for manipulation and none requires two hands for large size. Similarly if all of the components in the subassembly require one hand with grasping aids then the subassembly is assumed to share this characteristic. In this case, if one component requires special tools for manipulation then this characteristic is used for the entire subassembly. If special tools are not needed but standard tools are for a single component the subassembly is assigned this property. Otherwise, it is assumed that the subassembly requires tweezers for manipulation. If none of the above handling characteristics are present then it is assumed that only one hand is needed to manipulate the subassembly.

4.4 WASTED WEIGHT CALCULATIONS

Wasted weight calculations only use input information from node variable positions 0 and 1, material type and weight. At any point in the disassembly process components may be separated into any number of subassemblies. When wasted weight is calculated each one of these subassemblies is checked for material homogeneity. The wasted weight parameter would have a value of zero at the outset of the disassembly simulation meaning zero percent of the components by weight are in subgroups containing more than one material type. If all of the nodes in a subassembly have the

same value for variable 0 (composed of the same material), then they do not make a contribution to wasted weight since they are now in a state which allows them to be recycled. Otherwise the weight of every component in the subgroup is added together. Once each of the subassemblies is analyzed in this way the total wasted weight is divided by the total weight of all of the product's components and the wasted weight percentage is the result.

Chapter 5. The Disassembly Tree and Tree Evaluation

The introductory chapter of this thesis mentioned the concept of the disassembly tree. The disassembly tree is a structure which is used to represent all of the possible options which can lead a product to be completely separated into its constituent components. The end result of some number of disassembly operations is represented by a node or candidate in the disassembly tree.

5.1 FORMING THE TREE

Let us use the Toro leaf blower as an example. The completely assembled product is the first level of the disassembly tree. Assuming that the product begins in an orientation where access to the housing screws is blocked by a table, there are initially two disassembly options. The first option is the removal of the nozzle from the left half of the housing. The second option is the removal of the nozzle from the right half of the housing. These options lead to two identical nodes which depict the nozzle completely removed from the other components. These two paths are shown in Figure 5.1 and highlighted in green. At this point the only other options available in each branch are the removal of the housing screws however these cannot be removed until the assembly is flipped over. The flip rule application is highlighted in red. After this is done any one of the nine screws can be removed. The nine applications of a screw removal rule in each main branch of the tree are highlighted in blue. This process of evaluating the disassembly options at each node builds the disassembly tree.

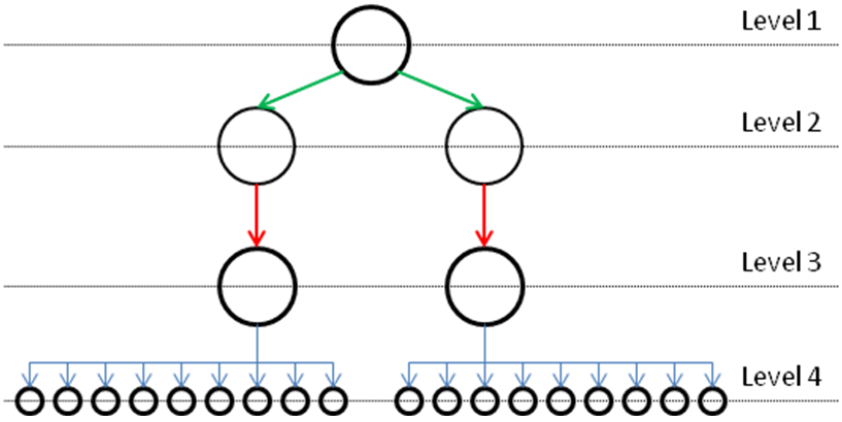

Figure 5.1: Formation of the Toro disassembly tree.

5.2 Candidate Evaluation and Terminating Conditions

In most disassembly trees, following any of the tree's branches to the lowest possible level reveals a candidate which represents a fully disassembled product. This candidate consists of each of the product's components, no longer attached to any others. With the goal of product recycling in mind and the provision of sufficient component material information the disassembly tree takes on a different appearance. The last node in any path down the tree consists of groups of components where each group is materially homogeneous. Using the two criteria of disassembly time and wasted weight each candidate in the tree can be evaluated. Once a candidate is determined to have a wasted weight value of zero there is no need to continue disassembly. A wasted weight value of zero is the terminating condition of the disassembly simulations conducted in this project.

5.3 Tree Search Output: The Pareto Set

As each node in the tree is evaluated the main purpose of the simulation is to find the candidate which simultaneously minimizes both disassembly time and wasted weight. However as the tree is searched there are other candidates between the product's

complete configuration and this optimal candidate which are worthy of note. These candidates are grouped together into what is called a Pareto set. Each candidate in the Pareto set is unique in the fact that no other node in the tree exists with lower values of both evaluation parameters. For each Pareto candidate if one of the two parameters is lowered by any amount no candidate with a lower value of the other parameter exists. It is said this candidate dominates all other nodes in the tree which possess equal values for either one of the two parameters. There are two noteworthy points in every Pareto set related to disassembly trees using these two parameters. The first point is the fully assembled product. With a disassembly time value of zero and a wasted weight value of one, the wasted weight cannot be decreased without increasing the disassembly time. Similarly the optimal point in the disassembly sequence possesses a wasted weight value of zero and the lowest possible disassembly time at which a wasted weight value of zero is achieved. No point in the disassembly tree possesses a lower disassembly time with an optimal value for wasted weight. Realistically all Pareto candidates represent various stages in the disassembly process. For example if the goal of the disassembly process was to recycle a certain percentage of material not equal to 100, the Pareto set would reveal the best candidate that would achieve this goal.

5.4 SEARCHING THE TREE

Functions can be hard-coded in C# to search the tree using any variety of techniques. It is within these programmable functions that terminating conditions, evaluation parameters and desired output information can be specified. A variety of search functions were written to search the tree and output a correctly representative Pareto set. Two factors to consider were the computational time of the search and the completeness of the Pareto set. A disassembly simulation which takes too long is not very

useful but neither is one which sacrifices completeness for a decrease in computational time. If too little of the tree is searched then some potential Pareto candidates may be missed.

5.4.1 Depth-First Search

Initial attempts to derive a complete Pareto set involved an exhaustive depth-first search. This search method searched the entire disassembly tree which would ensure that a complete Pareto set would be formed. The search worked by following the left-most branch of the tree from the first candidate to the last. Once each node in this branch is fully evaluated the path resulting from the latest possible point of divergence from the first branch is searched. This process continues until the search sweeps across the entire tree. Figure 5.2 shows the order of a depth-first search. The red path is searched first and the blue path is second.

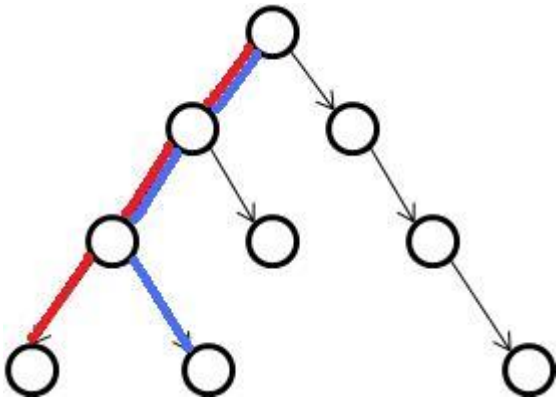

Figure 5.2: A simple depth-first search.

While the depth-first search functioned correctly and resulted in a complete Pareto set for the Toro leaf blower, the search took an unacceptably long amount of time to finish. On a desktop computer with an Intel Core 2 Duo CPU with a 3 gigahertz processor and 3.23 gigabytes of RAM, the search took 10 days, 16 minutes and 32.28

seconds. However the search did prove to be useful in some respects. Since it is an exhaustive search method the Pareto set is guaranteed to be complete and can be compared to Pareto sets output by non-exhaustive search methods to confirm their correctness.

One problem which was discovered from the first exhaustive search was that the branching factor of the disassembly tree was unnecessarily large. The fourth level in the tree consisted of eighteen nodes. These nodes resulted from two main tree branches which lead to identical subassemblies in which nine identical housing screws were the only disassembly options. Before the alteration of the thread constraint removal rules to include the simultaneous removal of identical components the next level in the tree had 144 nodes. If screws are removed one at a time then the removal of each screw creates eight new candidates since there are still eight screws remaining. The next level would consist of 1008 candidates (seven times 144). One can see that the tree was growing unnecessarily large very quickly. The simultaneous thread removal concept was developed to combat this problem. After this change was made, each of the eighteen candidates in the fourth level of the tree gave birth to one candidate at the fifth level, each of which contained no housing screws that were still attached.

A second change was an extension of this same concept that would be applicable to operations which were not thread-related. This addresses an issue in graph theory called confluence (Ehrig, 1999). Confluence refers to situations in which two different paths which would ordinarily lead to two identical nodes are combined to converge on one node. This concept is illustrated below in Figure 5.3. In the example depicted the application of two rules, a and b, leads to two different nodes, 3 and 5, which are indicated in red. However if 3 and 5 represent identical subassembly configurations then

the graph could be simplified by combining these two nodes into one. A function was written to attempt to apply this concept to a disassembly tree.

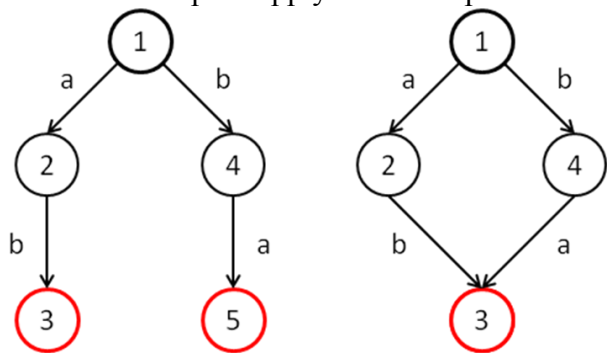

Figure 5.3: Confluence applied to a simple graph.

Each time a candidate's children were generated the disassembly time, wasted weight and last rule applied were compared. If all three of these were the same, one of the two children was designated a clone and removed from further consideration. Therefore, every candidate in the tree which was a derivative of this clone was also eliminated. In the Toro graph this would eliminate one of the two main branches of the tree resulting from the removal of the nozzle from each half of the housing. However as much time as these two alterations saved, both changes were implemented prior to the ten day search. The changes were actually put into effect when a depth-first search was allowed to run for fourteen days and had not yet searched one-eighteenth of the tree. The depth-first search was abandoned for reasons owing to computational inefficiency.

Various methods were investigated in an attempt to surmount the shortcomings of the depth-first search. These included another exhaustive method, breadth-first search, which was also computationally inefficient; and the first attempt at an informed search, uniform-cost search, which did not produce a complete Pareto set.

5.4.2 A* Search

After the failure of the first informed search method a second was tried which used both disassembly time and wasted weight to guide the search path. First, a combined function which took the summation of some proportion of the two parameters into account was formed. In the equation below A and B are weights chosen to change the effect that each parameter has on the function.

$$f = A \cdot g\,[disassembly\ time, seconds] + B \cdot h\,[wasted\ weight]$$

A* forms a list of candidates ordered from the lowest to the highest value of *f*. When the first candidate in the list is evaluated its children are generated and they are in turn placed in the list in the correct positions.

It was soon realized that with equal weighting, *g* dominated the right-hand side of the equation since *h* only varies between zero and one. In an attempt to attain a similar range of values for *g*, a random search was written. This search would run a disassembly sequence down a random branch of the tree, for a user-defined number of iterations. The average of the total disassembly times was taken and the reciprocal of this value was used as the weight on *g*. This ensured that the first parameter in the equation varied between zero and some number very close to one.

While A* greatly reduced the computational time (from ten days to less than one minute in some cases) the attempts to both normalize *g* and vary the weights on both *A* and *B* did not produce the desired results. The search process terminated as soon as the optimal point was found however this did not allow enough exploration of the tree to find a complete set of Pareto candidates. Figure 5.4 shows a comparison of the Pareto sets produced up to this point. The blue set of points, which is closer to the origin than the others, represents the depth-first search derived, complete Pareto set. Its accuracy is

confirmed by the fact that the candidates that it contains are more Pareto-efficient, i.e. they dominate, the candidates contained by the other sets.

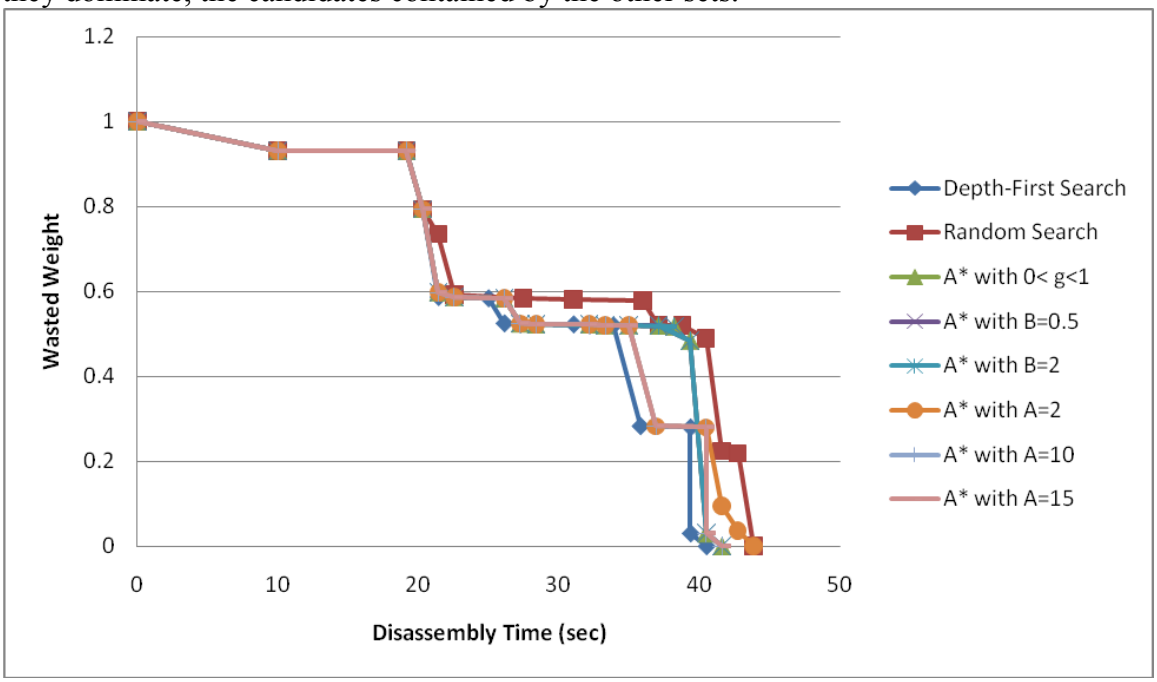

Figure 5.4: Comparison of Pareto sets produced by depth-first search to A* and random search.

5.4.3 Iterative-Deepening A*

Iterative-deepening A* (IDA*) performs multiple exhaustive searches with each of these searches at a progressively larger maximum depth (Kork, 1985). This means that the tree is only searched to a certain depth and then another user-defined parameter, epsilon (ε), defines the size of the difference between the maximum search depth from one iteration to the next. This search method ensures that the tree is only searched as far down as is needed to find the optimal point. Continuing the search across the tree at this depth ensures that the Pareto set is formed correctly. Results from multiple IDA* searches were encouraging. Using various values of epsilon the process was able to both greatly decrease computational time and produce the correct Pareto sets. Figure 5.5

shows that the Pareto sets produced by IDA* are complete while Table 5.1 shows the decreases in computational time.

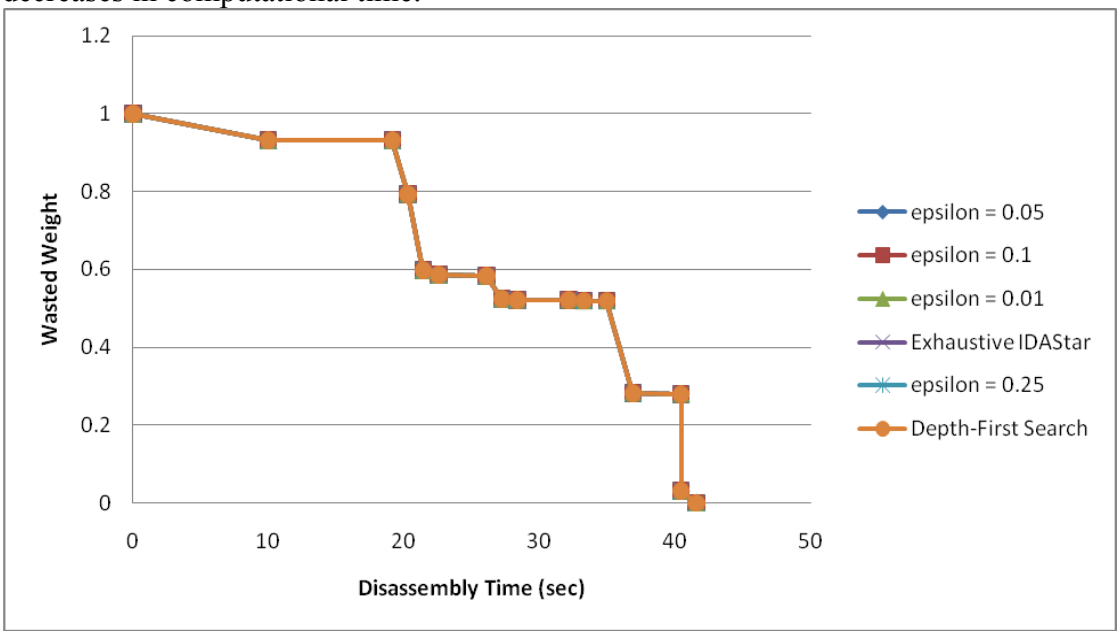

Figure 5.5: Comparison of Pareto sets produced by depth-first search to IDA*.

IDA* Results	
Epsilon	*Simulation time (dd:hh:mm:ss)*
0.01	00:00:23:23.7
0.05	00:00:15:25.2
0.1	00:01:01:20.0
0.25	00:00:23:39.6
N/A (Exhaustive search)	00:02:22:17.0
Depth-First Search Results	
-	10:00:16:32.38

Table 5.1: Comparison between IDA* and depth-first search computational times.

While IDA* proved to be both efficient and complete in the case of the Toro leaf blower the increased complexity, and therefore larger disassembly tree, of the Troy-Bilt model showed that IDA* was too time and memory intensive to be practical. It was also

discovered that if a candidate's graph possessed too many arcs, rule recognition and application were much slower than usual. This was especially true for the Troy-Bilt model. In its disassembly simulation the Troy-Bilt graph contained up to 332 arcs compared to a maximum of only 78 for the Toro model. At this point the concept of separate module graphs had not been created. Until this point the simple representation and individual module components were on the same graph and arcs with the label of "module" connected the module components to the module node. Rather than importing the module graph, the module rule removed both the module node and arcs when rule set #3 was entered. While implementation of the current version of the module rule and representation drastically decreased the number of arcs in all candidate graphs, the decrease was not enough to overcome the drop in rule recognition and application rates. In addition, a new problem involving memory exhaustion was encountered. The search method was simply storing too much information about previously investigated candidates for the computer's memory to store.

The main problem with IDA* was that it did not know the depth at which the optimal candidate existed until performing multiple exhaustive, depth-first searches. For a tree as expansive as that of the Troy-Bilt model, this was a critical failure. The solution to this problem was a combination of three of the search techniques which had already been investigated as well as the creation of an additional function to limit the size of the disassembly tree.

5.4.4 Combined Search Method

Three search methods are combined to create the solution to all of the issues that had been encountered: random search, A* and depth-first search. Along with these three searches two branching-factor decreasing functions are implemented. The first function

had been created earlier to handle the creation of identical children. The second function stores *g*, *h* and most recent rule applied information about every candidate evaluated up to that point. New candidates are not only compared with each other but also with every candidate that has come before. This leads to further reduction in the tree's branching factor.

The search now occurs in three steps:

1. A user-defined number of random searches are performed to find a normalizing factor for the disassembly time parameter.
2. The normalizing factor in step one is fed into an A* search which analyzes the tree until the candidate with the lowest possible value for disassembly time where wasted weight equals zero is found.
3. The minimum optimal depth from step 2 is fed into a depth-first search which analyzes the entire tree down to this level.

The above sequence produces complete Pareto sets while also completing a full simulation for the Toro leaf blower in less than five minutes and the Troy-Bilt leaf blower in two and a half hours.

Chapter 6. Results and Discussion

The primary means of presenting simulation analysis results is in the form of a Pareto set. Each point in the Pareto set represents a point in a disassembly sequence. While each candidate in this set represents a state in which a product is progressively more disassembled than in the last, the Pareto set need not contain a set of points which belong to the same, specific disassembly sequence. Each candidate in the Pareto set has a full range of information attributed to it including, but not limited to, the important parameters of disassembly time up to that point and the wasted weight which that candidate's subassembly contains. Keeping in mind the definition of dominance with regards to Pareto optimality and the fact that the disassembly algorithm searches the tree to a certain depth before sweeping across, it is evident why these points may not belong to the same sequence. In fact, the Pareto set also need not contain a number of points equal to the number of disassembly steps. A comparison between the most time-efficient disassembly sequence and the Pareto frontier of the Toro leaf blower is shown in Figure 6.1. The Pareto set contains a total of 26 candidates while completely disassembling the product only requires 19 individual operations.

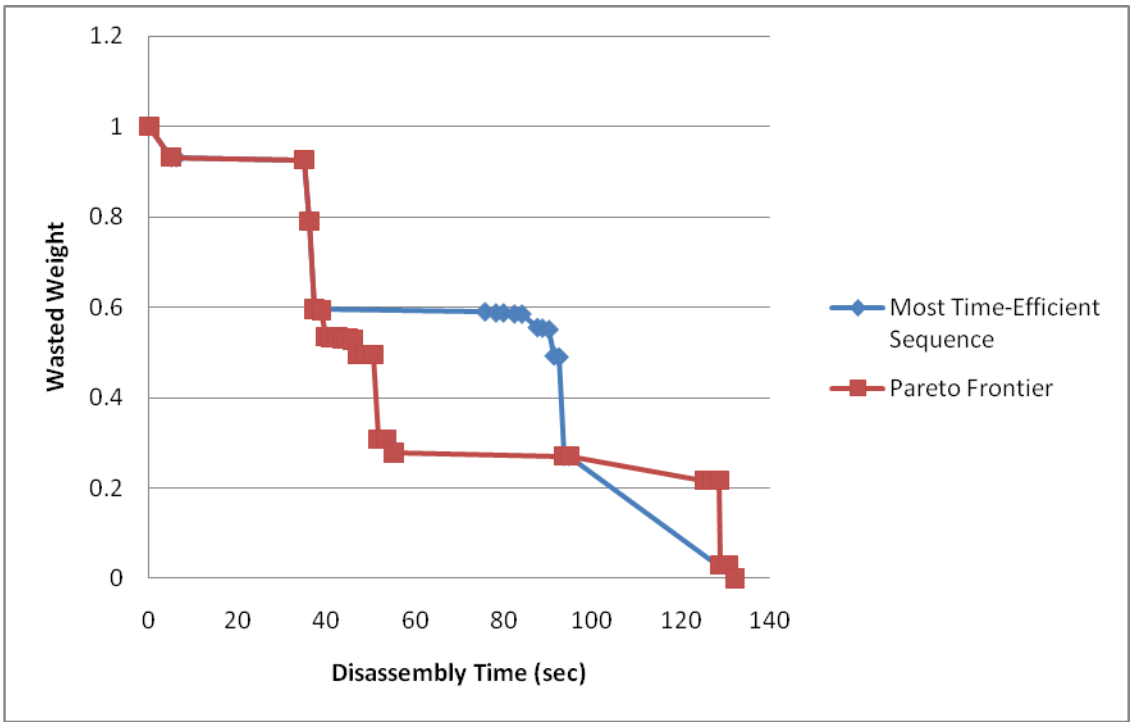

Figure 6.1: Most time-efficient sequence and Pareto frontier of the Toro leaf blower.

Additionally the "recipe" of each candidate is provided. The recipe is the list of disassembly operations which leads the fully disassembled product to attain a specific candidate's component configuration. For the Toro leaf blower the recipe of the most time-efficient complete disassembly sequence is as follows:

- Remove the nozzle from the (left or right half of the) housing,
- Flip the remaining connected components to allow access to the housing screws,
- Unscrew the nine housing screws,
- Remove the left half of the housing from the remaining connected components,
- Remove the motor,
- At this point the module rule is applied, indicating that removal of the motor is to be undertaken,
- Remove the tape which is around the left wire coil,

- Unwrap the left wire coil from around the stator,
- Remove the first screw from the stator,
- Remove the washer,
- Remove the second screw from the stator,
- Remove the washer,
- Separate the upper rotor bearing and the large plastic supporting piece,
- Separate the lower metal stator support and the stator,
- Unscrew the lock nut on the bottom of the rotor shaft,
- Remove the impeller from the rotor shaft,
- Remove the lock nut from the rotor shaft,
- Separate the lower metal stator support and the rotor shaft,
- Remove the tape which is around the right wire coil,
- Unwrap the right wire coil from around the stator.

A useful means of comparing two products is in contrasting their Pareto frontiers. Figure 6.2 compares the Pareto frontiers of the two leaf blower models. In this figure and in the Pareto calculations the wasted weight is expressed as a percentage. It can be seen that the Toro model's time-optimal complete disassembly sequence (represented by the point which touches the y-axis) is reached in a shorter amount of time than that of the Troy-Bilt model. It is also interesting to contrast the progression of each product's disassembly. For example in the time taken to completely disassemble the Toro model, approximately 80% of the Troy-Bilt leaf blower can be recycled at most.

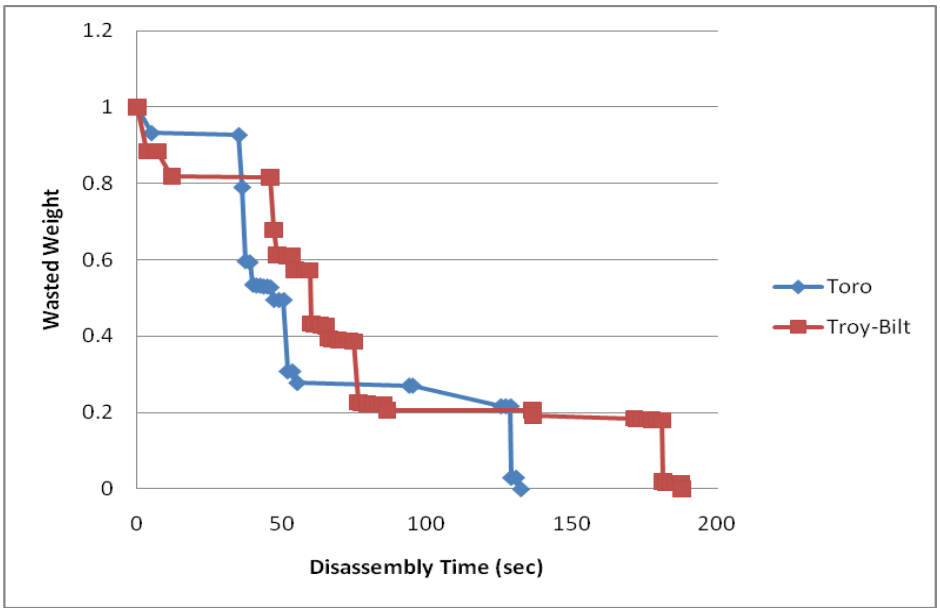

Figure 6.2: Pareto plots of the Toro and Troy-Bilt leaf blowers.

The results of simulations developed in this project present data that may be useful to a wide range of people. Moreover, while the primary focus remains on presenting data that is useful to the analysis of products which have reached their end-of-life, this same data could be useful during the design phase of a product. The analysis of specific points in the Pareto set yields information which could be used to redesign a product so that its disassembly is easier.. The nearly horizontal intervals in the plots shown above indicate disassembly operations which result in the liberation of a relatively small amount of material. The first long interval of this type in each plot results from the removal of nine housing screws. For example, the screws in the Troy-Bilt leaf blower account for 0.427% of the product's total weight. However their removal takes 29.35 seconds or 15.6% of the total disassembly time. While also keeping EPR requirements that govern product recycling and material usage in mind this analysis could be performed on products before they even reach production.

Otto and Wood specify a number of design guidelines which should be taken into account when designing a product while considering the processes involved in its eventual disassembly (Otto, 2001). Two of these guidelines suggest that as few fasteners should be used as possible and that any fasteners should be as similar to each other as possible. The effects of these guidelines are reflected in this example of the Troy-Bilt housing screws. Screw removal is the most time consuming type of operation in the disassembly of both models. The 29.35 second removal time of the housing screws accounts for the removal of all nine screws. These screws are grouped together because they connect common components and are of identical head type, removal method and thread length. If there were any variation in any of these characteristics then this long, nearly horizontal section would be broken up even further. Furthermore the potential inclusion of tool switching and acquisition time (see Section 7.4) would stress the importance of these design fundamentals. Other guidelines such as that involving removal direction, which suggests the use of "one disassembly direction to avoid reorientations," are also reflected by the results. Each of the leaf blowers only requires one flip of their assemblies however the requirement of repeated flip operations would negatively impact the performance of any product.

While the two leaf blower models are very similar they do not have identical weights and do not contain identical numbers of components. Therefore the concept of the best product to disassemble, with ease of disassembly related to the efficiency of the disassembly sequence, is not clearly defined. One way to compare the two sequences is to see which liberates material at a faster rate. Plots of wasted weight, expressed in pounds-force, versus disassembly time are shown in Figure 6.3.

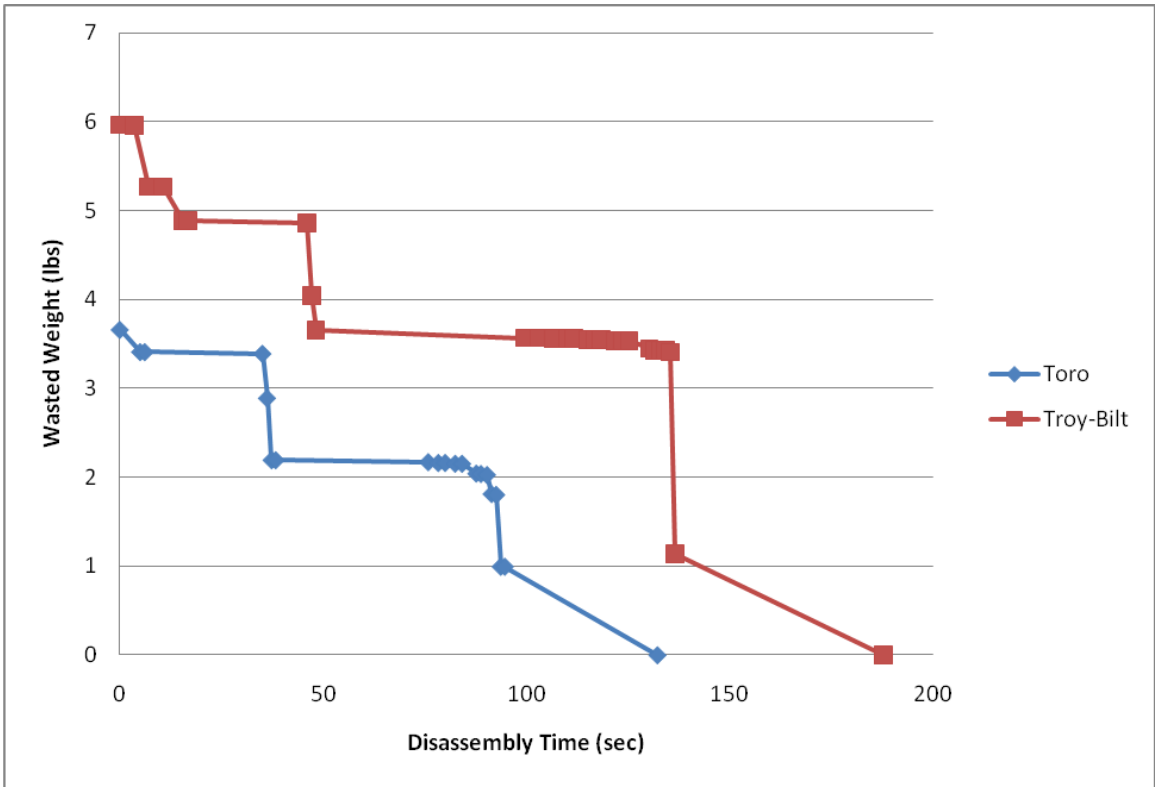

Figure 6.3: Comparison of weight liberation rates of two leaf blowers.

The "weight liberation rate *(w)*" of each leaf blower is defined as the total weight of the product divided by the total disassembly time and is calculated for each leaf blower in equations (6.1) and (6.2).

$$w_{Toro}\left(\frac{lbf}{s}\right) = \frac{3.661622227}{132.1382} = 0.027711\frac{lbf}{s} \qquad (6.1)$$

$$w_{Troy-Bilt}\left(\frac{lbf}{s}\right) = \frac{5.964617423}{187.79} = 0.031762\frac{lbf}{s} \qquad (6.2)$$

The above figures show that the even though the Troy-Bilt leaf blower takes more time to recycle than the Toro model, its disassembly sequence is more efficient. The most

efficient sequence of disassembly operations yields a larger amount of recyclable material per unit time. Analysis of the Pareto frontier of each product could lead to increases in these weight liberation rates by analyzing the horizontal sections of the frontiers which drag the rates down and making changes to lessen their effect. This weight liberation rate could also be used to compare products which are not as similar as these two. In fact the recyclability of any set of products could be compared using this metric. If EPR initiatives were to enforce the analysis of products using a metric of environmental suitability which takes figures such as this weight liberation rate into account, then consumers could be informed about the relative "greenness" of their potential purchases. Moreover, using this report's analysis method, designers and manufacturers could ascertain this same information in advance of production and also compare their product's performance to the performances of competing products.

Chapter 7. Recommendations for Future Work and Conclusions

While the current product representation and disassembly analysis methods produce the desired set of results there are various ways in which each of these can be improved. These suggested changes can be implemented at each level of the project: product representation, grammar rules, disassembly time calculations and disassembly tree analysis.

7.1 GRAPH FORMATION

At present, the most time-consuming part of the product representation and analysis process is the formation of the assembly graph. Some degree of familiarization with the terminology associated with component connection groups is required. This is a hurdle that would ideally be eliminated by automating the graph formation process. Many engineering programs that deal with the analysis of multi-component structures are able to import a model from CAD modeling software. Of course, these two programs most often have the same publisher and are designed to be able to communicate with each other. The information that is required for the formation of an assembly model in common CAD software, such as Pro/ENGINEER Wildfire 2.0 for example, is similar to the type required in the graph formation process used here. Certain surfaces on components are "mated" with each other and threaded surfaces can be specified as well. Ideally GraphSynth would be able to import an assembly and the pre-defined mating information would be used to build an assembly graph such as those shown in this report. Unfortunately the development of this capability was well outside the scope of this project. Future work performed in the pursuit of this ability would go a long way in increasing the ease of use of the disassembly analysis method presented here and decrease the requirements of the user.

7.2 GEOMETRIC INFORMATION

At present a three-dimensional Cartesian coordinate system is used to define direction. Of course, this does require some simplifications to be made to information related to the direction of components' connective relationships. In order to be completely accurate, angular offsets from the three Cartesian axes should also be taken into consideration. The CAD assembly importation suggested above would probably be able to take this type of information into account since this data is also supplied by users during the assembly modeling process. One potential location for the storage of angular information within the current graph structure would be the arc variable field. At present only certain types of connections use the variable field. Of these, only one set of connection types, the winding group, uses more than one position within the field. Alternatively the directional suffixes of many arc labels could be altered to include this type of information with combinations of angles and axes signifying offsets of certain angles of rotation from one axis to another. Of course, additional complexity would have to be added to grammar rules to take angles into account when determining applicable disassembly operations.

7.3 DISASSEMBLY TIME CALCULATIONS: LIMITATIONS IN SCOPE

While Boothroyd and Dewhurst's research has been a great benefit to the evaluation of disassembly operations the omission of some particular data could prove to be problematic in the analysis of other products.

For instance the functions used to calculate screw removal time are limited to only three head types. Information is provided on both human-powered and power-assisted removal concerning two of these three while only a human-powered function is provided for the other. Some of the products, outside of the two leaf blower models, which have been investigated so far, contain screws with torx heads. No information is

provided on this head type which means that completely accurate calculations could not be performed.

7.4 DISASSEMBLY TIME CALCULATIONS: MISSING INFORMATION

The absence of accurate adhesive removal information was mentioned earlier. Ideally a library which contains a list of adhesive types and their corresponding solvation times would be formed which could perform calculations related to adhesive removal.

While looking at the results of the disassembly simulations of the two leaf blower models, the short durations of the two sequences may be surprising. The shorter of the two sequences only takes two minutes and twelve seconds. Current disassembly calculations only take actual disassembly operations into account. However manual disassembly requires the switching of tools such as tweezers and screwdrivers. The time required to change from one tool to another would obviously have some effect on the results. The simultaneous screw removal function lessens some of the potential effects of tool acquisition time inclusion. Relaxation of this constraint combined with tool data would have an even more drastic effect.

The inclusion of this information would be fairly simple since the variables which are currently used in disassembly time calculations also provide information regarding the use of tools. For instance the choice of handling time variable values specifies whether or not optical magnification is needed to see a part or whether tweezers are needed to handle a component. Additionally, thread arc variables provide information regarding screwdriver use and type. The inclusion of this more detailed information could only lead to more accurate results.

7.5 DISASSEMBLY EVALUATION

While recycling and end-of-life usage of products is an environmental concern, their practice is often determined by economic considerations. The inclusion of certain data in the disassembly evaluation portion of this project would be able to take these considerations into account.

The information provided by intermediate Pareto candidates in particular would stand to benefit most from the inclusion of monetary data. Wasted weight is currently computed as either a percentage of the total weight or using exact weight in pounds, in effect treating all materials equally. In reality certain materials are worth much more than others and this information may be very useful to both designers and those responsible for disassembling products. Attaching certain values, which could be expressed in dollars per pound, to material types would provide this data. Combining a dollar value, rather than a wasted weight percentage, to candidates in the disassembly tree would also very likely lead to a far different Pareto frontier. Additional economic information could include labor costs which would instead be a function of the disassembly time up to that point.

7.6 CONCLUSIONS

This thesis presents a unique method of analyzing the disassembly problem. This includes the development of a new, graph-based product representation method. Unlike previous work in the field, grammar rules are used to simulate the product disassembly process. The results of this simulation are a more accurate approximation of real disassembly. Unlike existing methods, which use computational time as their guiding function, this method seeks a closer connection with the real issues faced when conducting manual disassembly. Through the formation of a disassembly tree, a thorough analysis of it and the inclusion of well-accepted empirical data (Boothroyd, 1989 and

Boothroyd, 2002) we are able to use disassembly time as the most important parameter in order to produce more useful results. The tree search method, which combines random, A* and depth-first searching techniques, combined with techniques to limit the unnecessary searching of multiple, identical nodes, is a unique approach in itself and has the potential to be used in other problems which face the issue of trees which are both deep and possess large branching factors. The use of Pareto frontiers and the parameters of disassembly time and wasted weight provide a means of comparing the performances of multiple products at numerous points in their respective sequences. The use of a newly created evaluative parameter, the weight liberation rate – which measures how much recyclable material is liberated during disassembly per unit time, provides an effective means of comparing the performances of different products undergoing complete disassembly.

The information that this method provides has the potential to be used in numerous ways. The Pareto frontier that each product possesses displays information which could be useful to designers and manufacturers. Some of the fundamentals of efficient design, such as those governing the use of fasteners, are clearly exhibited in the results. With the increasing influence of Extended Producer Responsibility initiatives on producers and their design practices, these results have the potential to provide important information to designers about the performances of their products during the end-of-life phase. With increasing concerns about the environment, the evaluation of products based on their recyclability and potential environmental impact is becoming increasingly important to producers and consumers alike. The use of the weight liberation rate as an environmentally-conscious grade on products has the potential to inform consumers about their product choices. The combination of more informed producers and consumers will lead to more efficient use of raw materials and a closer examination of the treatment

of discarded products; two of the most important environmental issues facing today's consumer-based society.

Appendix A. Rule Set #0: Fixer Rule Details

A.1 Single Fixer Rule

The single fixer rule simplifies the user's graph creation process. This rule is shown in Figure A.1.

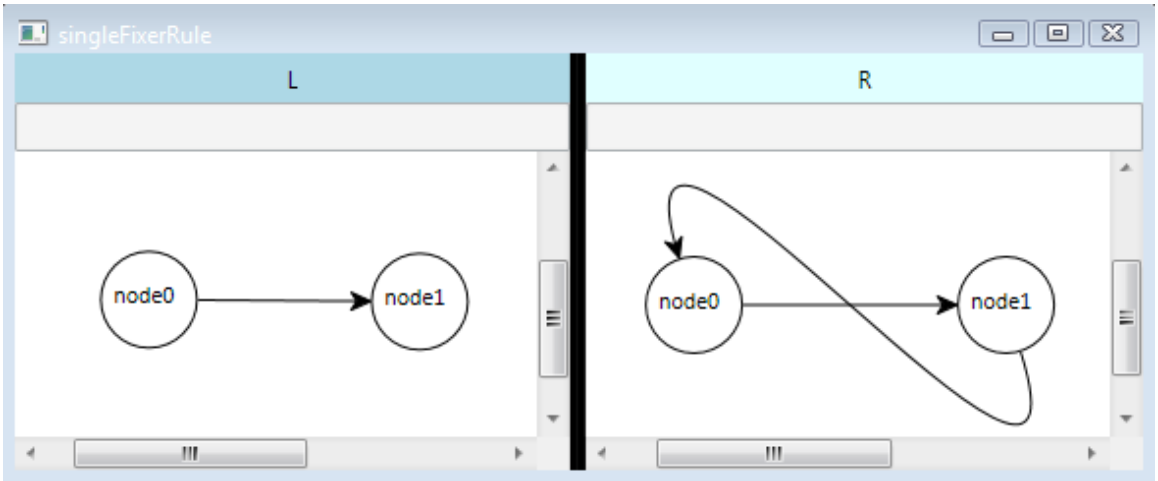

Figure A.1: Rule set #0: single fixer rule.

The left-hand side of the graph shows the first criterion for the rule's recognition. Since the arc between node0 and node1 does not specify a label, every single arc in the assembly graph passes the rule's first test. The second criterion involves a parametric recognition function. Each time an arc in the assembly graph is found it is checked for a label belonging to the rectangular constraint, radial constraint or press fit groups. If any of these labels are found then the function checks for the existence of an arc originating at node1 and terminating at node0 which also contains a label belonging to any of these three groups. If no arc is found then the second criterion is satisfied. The application of the rule also involves the subgraph depicted in R in addition to a parametric function. R

leads to the addition of an arc which originates at node1 and terminates at node0. The operation of the parametric application function is explained below:

1. Let arc a equal the arc originating at node0 and terminating at node1
2. Let arc b equal the arc originating at node1 and terminating at node0
3. If arc a contains labels belonging to the rectangular constraint group then rectangular constraint labels are added to arc b with identical Cartesian axial direction and opposite polarity
4. Step 3 is repeated for any labels belonging to the radial constraint group
5. Step 3 is repeated for any labels belonging to the press fit group

A.2 DOUBLE FIXER RULE

Situations in which a user chooses to draw both arcs between two components which share a rectangular, radial or press fit relationship are the scenarios which the double fixer rule was created for. If the user labels the arc going from node0 to node1 with rectConstraint+X and the arc from node1 to node0 with rectConstraint-Y then the labeling process is incomplete. Since the rule knows that a rectangular constraint arc between two components requires a second arc in the opposite direction and labeled with a constraint from the same group, along the same Cartesian axis but with opposite polarity, it will add the necessary labels to the existing arcs. The arc from node0 to node1 will receive an additional label of rectConstraint+Y and the other will receive a label of rectConstraint-X.

A.3 CASCADE FIXER RULE

The cascade fixer rule is unique in that both its recognition and application involve three separate nodes and six different arcs. The rule is shown in Figure A.2.

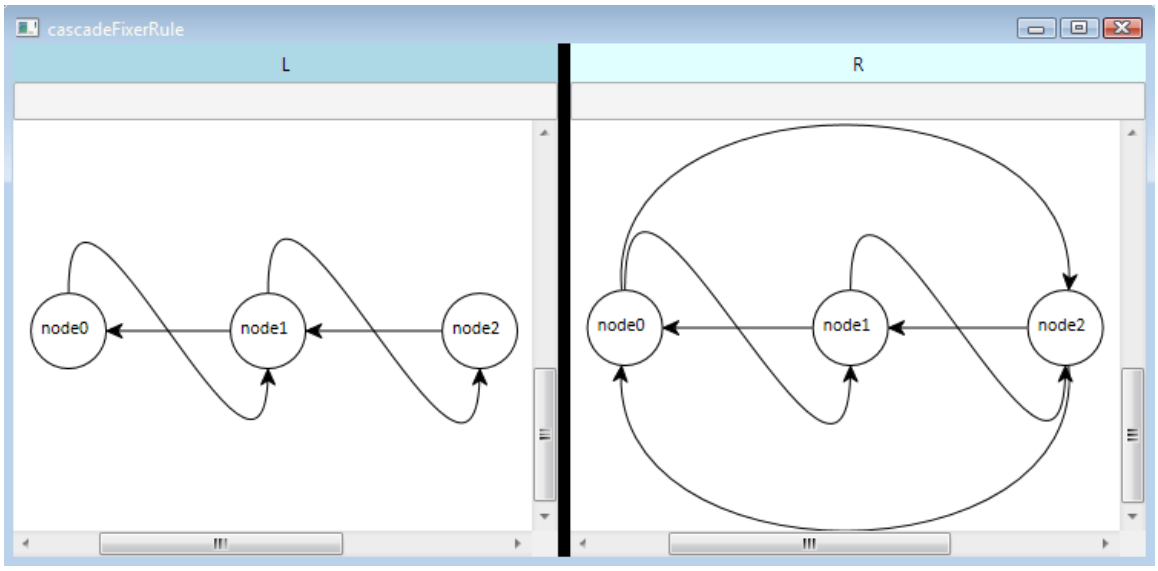

Figure A.2: Rule set #0: cascade fixer rule.

In each case the cascade fixer rule establishes a relationship between two components that would each become underconstrained in the event of the removal of a third component which they each share a connection with. As an example, and using the names in the figure above, if node0 is constrained by node1 by a rectangular constraint in the positive X direction and there is an identical relationship between node1 and node2, the removal of node1 would cause node0 to now be constrained by node2 in the positive X direction. However until node1 is removed, if possible, this constraint does not really exist. Therefore the cascade fixer rule draws an arc from node0 to node2 with the label virtualRectConstraint+X and an arc from node2 to node0 with the label virtualRectConstraint-X. The cascade fixer rule is not restricted to common rectangular constraint connections; it works in the same way for situations involving radial constraints and press fits. However for both of these scenarios the same group of virtual rectangular constraint labels is used. The rule also makes use of a property called induction in graph theory (Campbell, 2009). If a rule is "induced" then the arcs shown

between the three nodes in L and R are the only arcs allowed to exist between the three nodes which they are being matched with in the host graph. Therefore if there is already a connection between node0 and node2, the cascade fixer rule will not be recognized. This prevents the unnecessary creation of virtual rectangular constraint arcs between two nodes which may already have a regular rectangular, radial or press fit relationship.

Through comparison between Figure 2.10 and Figure A.3 one can see the effect of the application of fixer rules. Only after these rules are applied is the product representation graph constrained to the extent necessary to run an accurate disassembly simulation.

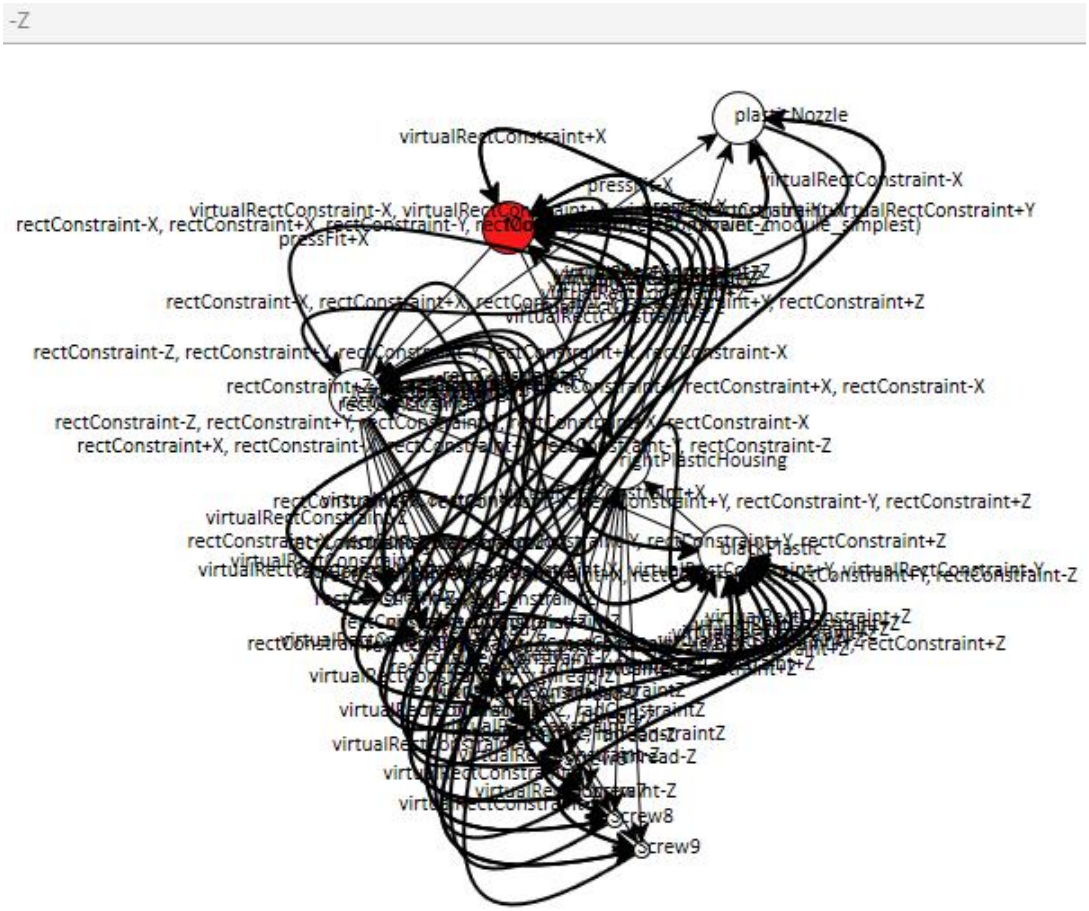

Figure A.3: Simple representation of the Toro leaf blower in GraphSynth after fixer rule application.

Appendix B. Rule Set #1: Disassembly Rule Details

B.1 Rectangular Constraint Removal

These grammar rules correspond to the removal of a component from the assembly when it is simply adjacent to other components and not fastened in any way, and when its removal can be achieved by translation in a single direction. Each one of the six rectangular constraints has a grammar rule which governs its removal. For instance, removeRectangularPXRule oversees removal of a rectConstraint+X labeled arc. The node from which node this arc originates is prohibited from being removed through simple translation in the positive X direction. If this arc is to be broken, i.e. removed from the host graph, then the component in question must be unblocked in the negative X direction. In the top left-hand corner of Figure B.1 the phrase "AND NOT (-X)" appears. This means that if the host graph carries a global label of –X, meaning the designated –X surface of the product rests on an immovable surface, the rule cannot be applied and node0 cannot be translated out of the assembly in the –X direction.

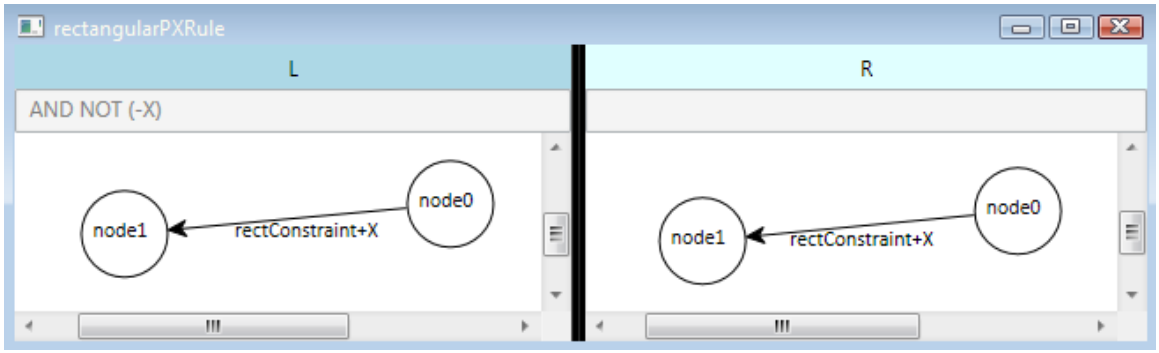

Figure B.1: The grammar rule governing removal of rectConstraint+X-labeled arcs.

The parametric recognition function for this particular rule operates as follows:

1. Let arc a equal the arc labeled with rectConstraint+X.
2. Let node n equal the node from which the arc originates.

3. If any arc originating from node n carries the labels rectConstraint-X, virtualRectConstraint-X, radConstraintY or radConstraintZ then the component cannot be removed.
4. If any of the arcs originating from node n does not carry a rectangular, radial or virtual rectangular constraint label then the component cannot be removed.
5. If any of the arcs terminating at node n carries a label from the thread group then the component cannot be removed.

Item 3 in the list above ensures that the component is free to be translated out of the assembly in the negative X direction. Item 4 ensures that it is not fastened to any other components; examples of this type of connection are an adhesive bond or a press-fitted connection. Item 5 ensures that the component does not have a threaded connection to another component.

Since the removal of a rectangular constraint implies complete removal from the assembly then once it is removed the component must not have a relationship with any other nodes in the graph. The parametric application function deletes all other existing connections that the node may possess.

B.2 RADIAL CONSTRAINT REMOVAL

Radial constraint removal is very similar to rectangular constraint removal. The only physical difference is that it has been determined that the component from which the radial constraint arc originates is on a shaft. Therefore the removal of this constraint would imply the translation of the component along the axial direction of the shaft and out of the assembly. The conditions for removal are the same as rectangular constraints only in this case the component must not have a radial relationship with any other components along either of the other two axial directions and it must not be blocked in

both the positive and negative direction by a combination of regular and virtual rectangular constraints in the Cartesian direction parallel to the shaft. The parametric application function also ensures the removal of any connections to other components.

B.3 THREADED FASTENER REMOVAL

The group of rules which govern the removal of constraints in the thread group equate to unscrewing a component which is mated with another along adjacent threaded surfaces. The parametric recognition functions for these rules are very simple. The directional suffix for each constraint in the thread group is the direction of removal of the component at which the thread arc terminates. The parametric function simply verifies that this component is not constrained by a rectangular, virtual rectangular or press fit constraint in this same direction and also that it is not constrained by a radial constraint which limits its movement along the Cartesian axis parallel to its direction of removal. Additionally a negating label ensures that the direction of thread removal is not blocked by an immovable surface.

The parametric function for application is much more complex. Threaded fastener removal is the only connection severance that can be applied to multiple, individual components simultaneously. The reasoning behind this originates from attempts to limit the branching factor of the disassembly tree during simulations (see Chapter 5: The Disassembly Tree and Tree Evaluation). In a real disassembly process if multiple, identical screws are used to attach two components it would not make much sense to remove half of the screws, move on to the removal of other components and then return to the other half of and remove them later on. Therefore the application function first analyzes the component whose threaded constraint is to be removed. If there are other screws which are connected to the same components as the original screw in the same

manner then these are added to a list of screws to be removed simultaneously. Finally, all of the components in this list are removed from the assembly.

B.4 Press Fit and Rivet Removal

The press fit parametric application function is very similar to that of threaded constraints. However since the directional suffixes on threaded constraints correspond to the direction of removal and those of press fit constraints to the direction of insertion, there are slight differences. In the end the function simply checks to make sure that the component in question is unblocked in the direction of removal. Again, a negating label ensures that the direction of removal is not blocked by an immovable surface.

Like threaded constraints there is an extra level of complexity to the application of press fit removal rules. An example which necessitates this extra consideration is the nozzle attached onto the outer housing of either of the two leaf blower models. The outer housing of each leaf blower consists of two halves, meaning the nozzle is press-fitted onto two components simultaneously. The parametric function checks for conditions such as these and ensures that if one of the press-fitted relationships is severed then the other is as well.

The recognition and application of rivet removal rules operate in exactly the same fashion and apply to connections in the rivet group rather than the press fit group.

B.5 Adhesive Removal

Adhesive removal is very simple and does not involve parametric functions. Recognition and application are both handled by L and R as shown in Figure B.2.

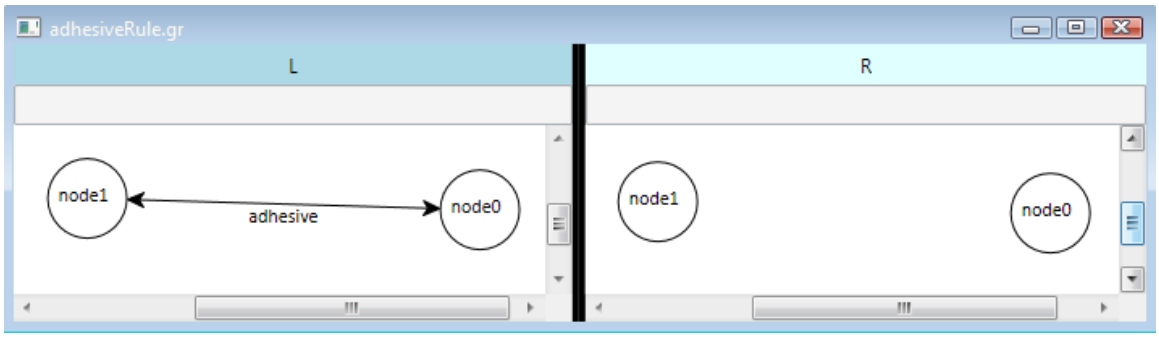

Figure B.2: The grammar rule governing removal of adhesive-labeled arcs.

B.6 WINDING REMOVAL

The removal of winding arcs is also very simple. The parametric recognition rule simply checks to ensure that no adhesive bonds the coiled component to another while the application rule simply severs its connections.

Appendix C. Rule Set #2: Flip Rule Details

Rule set #2 consists of six rules. Only groups of components whose rotation would actually serve a purpose with regards to additional disassembly are rotated. Since any one of the Cartesian directions may be blocked by an immovable surface there is one flip rule for each potential 180 degree rotation. For example, the rule called flipPXtoNXRule is only used if the global variable is +X. Just as a negating label specifies global labels whose presence prevents the recognition of a rule, LHS (standing for left-hand side) labels specify global labels which must be present for a rule to be recognized. The left and right-hand sides of each flip rule simply consist of a single node. This ensures that every node in the host graph is checked by a parametric recognition function. If the flipPXtoNXRule checks a node and sees that one of its outgoing arcs carries the thread+X label then it knows that removal of this constraint cannot be undertaken due to the orientation of the assembly. Similar checks are performed for arcs belonging to other constraint groups.

The node identified by the recognition function is then used by the application function. First a subgroup of nodes is formed consisting of all of the nodes to which this recognized node shares a relationship. This relationship can either be a direct connection or a common connection with another node. This subgroup is formed so that only collections of components which will benefit from being rotated undergo this action. The rule then flips the polarity of the directional suffixes of every arc label which matches with the rule's Cartesian axis. For example the flipPXtoNXRule changes all labels from thread+X to thread-X or rectConstraint-X to rectConstraint+X and so on. Since the global label has not been altered a screw which previously could not be removed in the positive X direction has now had its label changed to thread-X meaning it can now be removed.

Each of the six flip rules performs in this manner. If one of the flip rules is applied then this is a signal that further disassembly is possible and the simulation returns to the disassembly rule set. Otherwise, the module rule set is entered.

Appendix D. Rule Set #3: Module Rule Details

The module rule uses information from various places in its parametric application function to ensure that this further disassembly occurs. Once the module rule has been identified the directory of the assembly graph file is taken from the graph's "name" property. The requirement of the input of this information in the presence of modules was mentioned in Section 2.2. Then the filename of the module's assembly graph is taken from the module node's second label. The function then opens the module assembly graph file and copies all of its nodes and arcs into assembly graph of the product's simple representation. The simulation is then returned to the fixer rule set and continues the process from there. An example of the effects of the module rule is shown below. These figures relate to the application of the module rule for the Toro leaf blower. The separate assembly graphs of the entire product and the collection of module components are shown in Figure 2.10 and Figure 2.11 respectively. Notice that in Figure D.1 the components belonging to the assembly's simple representation are fully disconnected on the left-hand side of the image. The components of the right-hand side comprise the motor and the additional arcs seen in this image compared to the one shown before are the result of the application of fixer rules to the newly added nodes. Also take note of the absence of the node which carried the "module" and "leaf_blower_module_simplest" labels (and was highlighted in red) from the earlier figure. This node represented the motor as a whole and since the motor's individual components have been added to the host graph the module node has been removed.

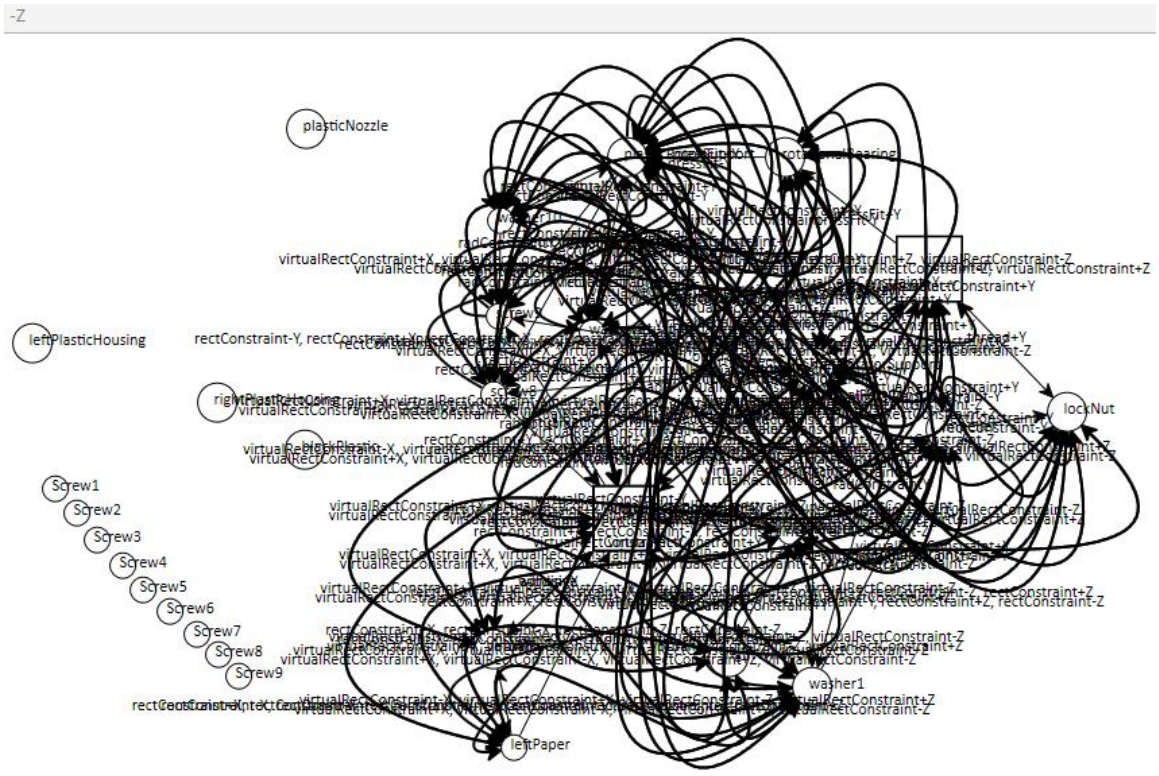

Figure D.1: Toro leaf blower after the application of the module rule and a second iteration of the fixer rule set.

At this point a simulation can be run which represents the disassembly of a full product however without the inclusion of information about the product's components no calculations relating to disassembly time and wasted weight can be performed. Specific information must be supplied to increase the accuracy of the calculations and specific locations are designated for their entry into the model.

References

Boothroyd, G. and Dewhurst, P., *Product Design for Assembly*, 1989 (Wakefield: Boothroyd Dewhurst, Inc.)

Boothroyd, G., Dewhurst, P. and Knight, W., *Product Design for Manufacture and Assembly*, 2002 (New York: Marcel Dekker)

Campbell, M., *A Graph Grammar Methodology for Generative Systems*, April 2009, http://hdl.handle.net/2152/6258

Campbell, M. and Hasan, A., Design Evaluation Method for the Disassembly of Electronic Equipment, in *International Conference on Engineering Design* (ICED) 03, Stockholm, August 2003.

Censor, Y., Pareto Optimality in Multiobjective Problems, in *Applied Mathematics and Optimization*, Vol. 4, No. 1, 1977 (New York: Springer New York).

Chen, S., Oliver, J.H., Chou, S. and Chen, L., Parallel Disassembly by Onion Peeling, in *Journal of Mechanical Design*, Vol. 119, June 1997.

Ehrig, H., Rozengerg, G., Engels, G. and Kreowski, H.J., *The Handbook on Graph Grammars and Computing* by Graph Transformation: Applications, Languages and Tools, 1999 (United States: World Scientific Publishing Company)

Environmental Leader, *53% Of Consumers Prefer To Buy From Company With Green Rep*, 2 October 2007, http://www.environmentalleader.com/2007/10/02/53-of-consumers-prefer-to-buy-from-companies-with-green-rep/

Extended Producer Responsibility Laws as of July 2009: http://www.productstewardship.us/displaycommon.cfm?an=1&subarticlenbr=280

GraphSynth website: http://www.me.utexas.edu/~adl/graphsynth/

Gross, D. and David, E., The Tao of Junk, *Newsweek Web Exclusive*, 5 September 2007, www.newsweek.com/id/39771

Gutowski, T., Dahmus, J., Albino, D. and Branham, M. Bayesian, Material Separation Model with Applications to Recycling, in IEEE *International Symposium on Electronics and the Environment*, Orlando, May 2007.

Homem de Mello, L. S. and Sanderson, A. C., AND/OR Graph Representation of Assembly Plans, in *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 2, April 1990.

Kork, R. E., Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, in *Artificial Intelligence*, No. 27, 1985.

Otto, K. and Wood, K., *Product Design: Techniques in Reverse Engineering and New Product Development*, 2001 (New Jersey: Prentice Hall, Inc.)

Pro/ENGINEER Wildfire 2.0 webpage: http://www.ptc.com/partners/hardware/current/support/proewf2.htm

Russel, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, 2002 (New Jersey: Prentice Hall, Inc.)

Srinivasan, H. and Gadh, R., Efficient Geometric Disassembly of Multiple Components from an Assembly Using Wave Propagation, in *Journal of Mechanical Design*, Vol. 122, June 2000.

Toro – Interactive Product Manuals: Power Sweep (51586): http://216.213.143.2/ProAuthor/ManProc.cfm?&sk=623F3BE5%2D7D03%2D11DE%2D9262%2D00D0B789305D&&&ProcID=20812&ManualID=299&openMenu=20812

Troy-Bilt website: www.troybilt.com

Vita

David Ikechukwu Agu was born in Houston, Texas on May 17, 1985. The son of Dr. and Mrs. Vincent and Pamela Agu, David spent his childhood growing up in various countries including Congo-Brazzaville, South Africa and Switzerland. As an undergraduate he attended Stanford University in Palo Alto, California and graduated with a Bachelor of Science in Mechanical Engineering in June 2007. He enrolled in the Cockrell School of Engineering at The University of Texas at Austin in August 2007. At the graduate level he also majors in mechanical engineering with a concentration in Manufacturing and Design.

Permanent address: 4412 Avenue A Apt. 208, Austin, TX 78751

This thesis was typed by David Ikechukwu Agu.